UNIVERSITY OF GOTHENBURG

SOLWEIG – A climate design tool

Master of Science Thesis

DEEPAK JESWANI DEWAN

University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, July 2009

SOLWEIG – A climate design tool

DEEPAK JESWANI DEWAN

Examiner: ROGARDT HELDAL.

Supervisors: ROGARDT HELDAL, SOFIA THORSSON, FREDRIK LINDBERG.

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden July 2009

Introducción

El objetivo de este resumen es presentar y explicar de forma resumida toda la información relacionada con el proyecto "SOLWEIG – a climate design tool", una aplicación software gráfica y amigable que integra y ejecuta el modelo de software llamado SOLWEIG. SOLWEIG es un modelo de radiación que realiza estimaciones climatológicas (como por ejemplo estimar los patrones de sombreado diurnos) y analiza la compleja interacción entre el diseño urbano y el ambiente térmico. Fue desarrollado por el Grupo de Climatología Urbano en el Centro de Ciencias de la Tierra, Universidad de Gotemburgo, Suecia.

El modelo SOLWEIG ha demostrado ser una herramienta útil para aquellos investigadores y profesionales relacionados con la rama de estudio. Sin embargo, algunas restricciones, como ser un modelo ejecutado en la línea de comandos o la necesidad de comprar una licencia para poder ejecutarlo, hace que sea difícil compartirlo con otros usuarios. Estas limitaciones animaron a los investigadores del Grupo de Climatología Urbano a solicitar una herramienta gráfica y amigable que integrase el modelo SOLWEIG y solucionase estos problemas de distribución.

Para el desarrollo de la herramienta, tres requisitos se han tenido como objetivos: amigabilidad, extensibilidad y usabilidad. Mientras que la extensibilidad está dirigida a los futuros desarrolladores de la herramienta, de tal manera que el software sea fácil de ampliar y actualizar, la amigabilidad y usabilidad tienen el objetivo de garantizar una interfaz con buen aspecto, elegante y fácil de usar que animará a los usuarios a utilizarla más de una vez. Con el fin de medir el cumplimiento de estos tres requisitos, la herramienta ha sido probada y evaluada por usuarios normales e investigadores que están familiarizados con el modelo SOLWEIG.

La interfaz gráfica ha sido escrita en Java e integra el modelo SOLWEIG, que está escrito en MATLAB. Por lo tanto, una de las principales ventajas de esta herramienta es que ofrece una forma alternativa de utilizar una aplicación de MATLAB combinada con una potente interfaz gráfica.

Cabe mencionar que esta herramienta es útil para aquellos que están investigando en la materia de estudio, así como para los arquitectos y planificadores urbanos. La herramienta es gratuita y ejecuta el modelo SOLWEIG utilizando el compilador en tiempo de ejecución de MATLAB (MATLAB Compiler Runtime), que puede ser libremente distribuido junto con la interfaz gráfica.

Desarrollo del proyecto

En esta sección del resumen se van a explicar todos los pasos que se han seguido hasta obtener el producto final. En primer lugar, se introduce una breve descripción sobre el modelo SOLWEIG y sus aplicaciones. A continuación, se da un argumento a favor de la necesidad de desarrollar una interfaz gráfica amigable para el modelo descrito y la metodología de trabajo que se ha seguido para obtener dicho software, el cual debe cumplir los requisitos que en esta sección se exponen. Tras esta parte, se anuncia el lenguaje de programación que se ha elegido para construir la interfaz gráfica teniendo en cuenta las limitaciones que se encontraron y, además, se describe la forma en la que el modelo SOLWEIG se integra dentro de la interfaz. Acto seguido se comentan las características de la interfaz a modo de información técnica que explican y cubren los requisitos de la aplicación. Finalmente, se comentan brevemente los resultados obtenidos de haber sometido la interfaz a diferentes tests de pruebas ejecutados por un conjunto de usuarios.

El modelo SOLWEIG

SOLWEIG proviene del acrónimo SOlar and LongWave Environmental Irradiance Geometry, y es una solución software que calcula las variaciones espaciales de flujos de radiación tridimensionales y la temperatura media radiante (T_{mrt}) en complejos urbanos. Fue desarrollado por el Grupo de Climatología Urbano en el Centro de Ciencias de la Tierra, Universidad de Gotemburgo, Suecia.

La T_{mrt} es uno de los factores meteorológicos más importantes que rigen el equilibrio de energía humano y el confort térmico del hombre al aire libre. Se puede definir como:

“La suma de los flujos de radiación de onda corta y larga (tanto directos como reflejados) a la que el cuerpo humano está expuesto."

El modelo SOLWEIG basa sus orígenes desde una perspectiva de diseño urbano sostenible, con el objetivo de determinar cómo los flujos de radiación y la T_{mrt} pueden afectar a la salud y el bienestar de los seres humanos en un entorno urbano a lo largo de un día concreto.

El modelo está dirigido a los investigadores de la materia, arquitectos y planificadores urbanos.

SOLWEIG está escrito en el lenguaje de programación MATLAB. Esto implica un cierto número de ventajas para el objetivo de este modelo, ya que se requiere un alto grado de procesado de matrices e imágenes, requisito que MATLAB cubre perfectamente.

El modelo SOLWEIG puede ser aplicado para los siguientes objetivos:

- Realizar estimaciones climatológicas (patrones de sombra, flujos de radiación, temperatura media radiante, etc.) en los entornos urbanos.
- Analizar la interacción entre el diseño urbano y el ambiente térmico.
- Crear de mapas de confort térmico.
- Analizar los efectos del cambio climático en el entorno urbano.

Necesidad de desarrollar una interfaz gráfica amigable

El Grupo de Climatología Urbano de la Universidad de Gotemburgo demostró que SOLWEIG es un modelo de gran utilidad para aquellos usuarios relacionados con la rama de estudio, tales como investigadores, arquitectos y planificadores urbanos. Pronto surgió la necesidad de distribuir y compartir el modelo con más usuarios. Sin embargo, comprobaron que existían ciertas limitaciones en el modelo que hacía difícil su distribución y facilidad de uso, tales como:

- SOLWEIG es un modelo que usa la línea de comandos como interfaz. Esto significa que, con el fin de comunicarse con el software, el usuario tiene que teclear las órdenes y los datos de entrada en la línea de comandos para realizar las distintas tareas.
- No se proporcionan ventanas ni botones durante la simulación, objetos gráficos que hoy en día son típicos y muy comunes en toda aplicación destinada al usuario.
- El modelo opera de tal manera que va indicando al usuario la próxima entrada de datos que debe insertar. Se sigue un flujo secuencial y ordenado, dividido en etapas, antes de obtener los resultados y terminar la ejecución. De esta manera el usuario se ve obligado a reiniciar (y por lo tanto re-teclear) el modelo cada vez que lo quiera ejecutar y obtener nuevos resultados.
- Cualquier error al escribir o al cargar un archivo obliga al usuario a reiniciar toda la simulación de nuevo.

Estas limitaciones infieren que el modelo no es amigable, ya que no fomenta su reutilización de una forma fácil y cómoda.

Con el fin de compartir el modelo con futuros usuarios, es necesario proporcionar al menos una interfaz amigable que garantice la usabilidad, un buen rendimiento y, si es posible, un servicio de calidad aceptable; requisitos que pueden cumplirse mediante el desarrollo de una interfaz gráfica de usuario, que es capaz de resolver todas las limitaciones que una interfaz basada en línea de comandos manifiesta. Concretamente en este caso, con una interfaz gráfica de usuario, el usuario puede:

- Hacer uso de ventanas y botones con el fin de ejecutar las diferentes tareas, en lugar de teclear las órdenes y los datos de entrada.
- Romper el flujo secuencial y permitir recargar o volver a los datos de entrada de una etapa anterior.
- No finalizar el software después de mostrar los resultados, por lo que el usuario puede volver a ejecutarlo tantas veces como sea necesario sin necesidad de reiniciar la aplicación y recargar los datos.
- Cualquier error al cargar o seleccionar los datos no fuerza al usuario a reiniciar la aplicación.

Estas ventajas alientan la idea de desarrollar una interfaz tanto gráfica como amigable para lograr el principal objetivo del Grupo de Climatología Urbano: garantizar la productividad y el constante uso del software por parte de los usuarios, proporcionando una interfaz visual e intuitiva.

Metodología

Una vez comprobada la necesidad de desarrollar una interfaz gráfica y amigable para el modelo SOLWEIG, se establecieron las pautas para obtener dicho software de la manera más eficiente posible. Con el fin de explicar los distintos métodos que se han aplicado durante el proyecto, en primer lugar, es necesario agrupar las principales tareas que han implicado el proceso de desarrollo del software:

1) Comprender el modelo SOLWEIG y la teoría que la respalda.
2) Analizar el código fuente del modelo SOLWEIG para extraer los requisitos, funciones y formatos del software.
3) Actualizar el código fuente del modelo de tal manera que facilite su integración dentro de la interfaz gráfica.

4) Decidir el lenguaje de programación con el que construir la interfaz gráfica, teniendo en cuenta algunos requisitos importantes como: la integración de código, usabilidad, amigabilidad, extensibilidad (para las versiones futuras del modelo) y la calidad de los resultados.
5) Desarrollar la interfaz gráfica prestando especial atención en la amigabilidad y usabilidad, desde el punto de vista del usuario, y la extensibilidad, desde el punto de vista del programador.
6) Comprobar si la interfaz cumple los requisitos previamente mencionados.

Con el fin de resolver estas tareas, los siguientes métodos se han aplicado durante todo el desarrollo de proyecto:

a) Reuniones con los investigadores del Grupo de Climatología Urbano en el Centro de Ciencias de la Tierra (Universidad de Gotemburgo, Suecia) para resolver las tareas 1 y 4.
b) Reunión con el desarrollador del modelo SOLWEIG (del Grupo de Climatología Urbano en el Centro de Ciencias de la Tierra, Universidad de Gotemburgo, Suecia) para debatir y resolver la tareas 2 y 3. Mencionar que el desarrollo de casos de uso ha sido la vía de comunicación entre el desarrollador del modelo y la interfaz gráfica, pudiendo entender el problema y los requisitos de una forma gráfica y sencilla.
c) Desarrollar los correspondientes documentos de requisitos de usuario y sistema para discutir las tareas 4 y 5 y resolver la tarea 2.
d) Uso de la correspondiente plataforma de programación para resolver la tarea 5.
e) Recopilar las opiniones de un grupo de usuarios probadores del software para resolver la tarea 6.

Además, se han llevado a cabo reuniones y presentaciones semanales con los supervisores del proyecto con el objetivo de mantenerles actualizados con las novedades y progresos que acontecían en el proyecto durante la correspondiente semana.

El lenguaje de programación

El primer punto importante en relación al desarrollo de la interfaz gráfica fue tomar una decisión sobre el lenguaje de programación a utilizar. Dado que modelo SOLWEIG está escrito en MATLAB, la idea inicial fue la de adaptar la interfaz de línea de comandos del modelo a una interfaz gráfica. MATLAB ofrece un conjunto de cajas de herramientas que permiten al desarrollador crear aplicaciones basadas en ventanas, por lo que podría haber sido posible actualizar el modelo y adaptarlo a una nueva versión gráfica utilizando el propio lenguaje de programación MATLAB. Sin embargo, existen algunas limitaciones que recomendaron no optar por esta opción. Dichas limitaciones son las que se enumeran a continuación:

- MATLAB es un producto propiedad de The MathWorks, donde los usuarios dependen de un proveedor de productos y servicios (en este caso MathWorks). Esto significa que, a fin de ejecutar el modelo, el usuario final debe tener una licencia válida de MATLAB instalado en su ordenador. Esta licencia no es gratuita y podría provocar la negligencia del usuario de comprar una y, por lo tanto, de utilizar el software.
- El hecho de que existan bibliotecas gráficas más potentes, pertenecientes a otros lenguajes de programación, que ofrecen mayor cantidad de opciones y posibilidades a los desarrolladores para el cumplimiento de uno de los principales objetivos del proyecto: una interfaz amigable.

Una vez rechazada esta primera opción y debido a la limitaciones de MATLAB para exportar el código a un cierto lenguaje de programación, el lenguaje seleccionado fue Java. De entre todas las características que este lenguaje tiene, vale la pena mencionar los siguientes:

- Se trata de uno de los lenguajes de programación más conocidos y utilizados para desarrollar interfaces gráficas y páginas Web. Ofrece una biblioteca gráfica muy potente, llamada Swing, que contiene las características necesarias para obtener una buena interfaz amigable.
- Es gratuito y sólo requiere una máquina virtual (también gratuita) instalada en el ordenador del usuario final a fin de poder ejecutar cualquier aplicación Java.

Estas y otras ventajas hacían posible alcanzar los principales objetivos del proyecto y es por esto por lo que este lenguaje de programación fue el utilizado para el desarrollo de la interfaz.

En cuanto a la programación se refiere, se utilizó la plataforma NetBeans IDE 6.5 para el desarrollo de toda la interfaz gráfica, la cual está escrita bajo la versión 6 de Java.

Comunicación entre el modelo SOLWEIG y la interfaz gráfica

Tras tomar la decisión de no usar el propio lenguaje de programación MATLAB para desarrollar la interfaz gráfica, surgió la necesidad de vincular el modelo SOLWEIG (escrito en MATLAB) con el otro lenguaje de programación (Java). La primera idea fue la de reescribir el modelo SOLWEIG en Java, pero debido a algunos inconvenientes importantes, como por ejemplo carecer del potente procesado de matrices que MATLAB ofrece, esta opción fue rechazada.

Como MATLAB es un producto que tiene propietario, se hizo obligatorio comprobar las diferentes soluciones que el propio proveedor ofrece para este propósito. Se descubrió que un conjunto de herramientas y compiladores llamados MATLAB Compiler y MATLAB Builder proporcionaban la conversión de funciones escritas en MATLAB a una biblioteca de archivos que podría ser utilizada por los lenguajes C, C++, Excel, .NET o Java. Esta era la solución para vincular el modelo con alguno de los lenguajes propuestos por el proveedor, de entre los cuales los siguientes eran adecuados para el objetivo del proyecto:

- C++.
- .NET.
- Java.

Como se ha explicado anteriormente, el lenguaje de programación escogido fue Java. Por lo tanto, el correspondiente componente MATLAB que se utilizó fue el llamado MATLAB Builder JA, que agrupa a las funciones MATLAB que definen el modelo SOLWEIG en un único archivo de biblioteca Java.

Esta solución conlleva un pequeño inconveniente, que consiste en que el equipo donde la aplicación ha de instalarse debe utilizar un compilador en tiempo de ejecución llamado MCR (MATLAB Compiler Runtime), que hace funcionar correctamente las funciones MATLAB agrupadas en la biblioteca Java. Esto significa que, además de la interfaz gráfica, se debe instalar una aplicación adicional que quitará más tiempo al usuario y más recursos informáticos. Sin embargo, el MCR se obtiene con la herramienta MATLAB Compiler y puede ser distribuido libremente junto con los archivos de biblioteca Java generados por el propio MATLAB Compiler.

Una de las principales ventajas de esta solución es que el usuario tiene una forma alternativa de utilizar aplicaciones MATLAB junto con una potente biblioteca gráfica sin necesidad de comprar ninguna licencia o software adicional.

Requisitos de usuario y del sistema

Una vez conocido el lenguaje de programación y la forma de comunicar el modelo con la interfaz gráfica, el siguiente paso fue estudiar y extraer las diferentes etapas en las que se divide el modelo SOLWEIG en su versión original. Esta tarea se entendió como una forma de obtener los requisitos de usuario de cara a la aplicación final. Fue de relativa importancia el encontrar una separación del código en etapas que permitiese la organización del modelo de una forma lógica y fácil de seguir por parte del usuario.

Una vez extraídas las etapas, en formas de casos de uso, se procedió a la obtención de los requisitos del sistema. Para ello, se explotó cada caso de uso en su correspondiente conjunto de escenarios, que mostraba el comportamiento del sistema para cada posible interacción del usuario con el mismo.

Los requisitos del sistema se definieron teniendo en cuenta uno de los objetivos del proyecto: la usabilidad. Además, se aplicaron las siguientes características:

- Los botones están tratados de tal manera que guían al usuario durante el proceso en la mayoría de los casos.
- Los nombres de las ventanas y los mensajes de los diálogos están cuidadosamente seleccionados para que sean auto-explicativos y concluyentes.
- No hay forma alguna de llegar a un cuarto nivel de ventanas, partiendo siempre de la ventana principal, de modo que el usuario puede seguir fácilmente la tarea que esté realizando sin olvidar o no entender su objetivo.

Arquitectura y diseño del sistema

Los requisitos del sistema definían la funcionalidad de la aplicación final, pero no daban una clara idea acerca de la organización de los datos y la forma en que éstos se manejan internamente. Para lograr esto, se debía definir la arquitectura del sistema.

La arquitectura del sistema se definió teniendo en cuenta el objetivo más importante del proyecto desde el punto de vista del programador: la extensibilidad del código. Es más que probable que el modelo se actualice con bastante frecuencia y que se añadan más y nuevas funcionalidades en cada actualización. Por lo tanto, la interfaz gráfica debía estar preparada para ser fácilmente adaptada a estas futuras actualizaciones sin que ello implicase grandes y decisivos cambios en la misma.

Como requisito del Grupo de Climatología Urbano, era importante aislar el código del modelo del de la interfaz gráfica. El código del modelo tenía su versión en la interfaz en forma de biblioteca Java. Por lo tanto, la arquitectura del sistema debía separar los componentes gráficos de las referencias a la biblioteca Java que representa el modelo SOLWEIG.

En base a este requisito y teniendo en cuenta la extensibilidad del código, la arquitectura Modelo-Vista-Controlador fue la elegida para organizar la organización interna del sistema.

Con esta arquitectura, los datos que pertenecen al modelo SOLWEIG quedan almacenados y gestionados en el componente Modelo, mientras que la Vista contiene todo lo relativo a los componentes gráficos. El Controlador se encarga de unir ambos componentes principales. Las diferentes actualizaciones del modelo SOLWEIG pueden afectar a la interfaz de distintas maneras, pero no implicará grandes cambios en ella. Los posibles casos se enumeran a continuación:

- Una actualización en el modelo que no cambie ningún parámetro de entrada y salida en la correspondiente función MATLAB no implicará ningún cambio en la interfaz, ni siquiera en el componente Modelo.
- Una actualización en una función que cambie algún parámetro de entrada y/o salida implicará actualizar el Modelo, el Controlador y la rara vez Vista, pero en pequeña escala.

- Añadir una nueva función que implique la creación de una nueva etapa en el modelo implicará actualizar todos los componentes. La actualización del Modelo y el Controlador no implicará grandes cambios (sólo la adición de un nuevo sub-módulo en el Modelo y una referencia a él en el Controlador), mientras que la Vista dependerá de cómo esté diseñada la interfaz gráfica.

En lo que respecta al componente Vista, además de su diseño interno, fue también necesario establecer los requisitos de diseño para obtener una apariencia del sistema que cubriese el objetivo más importante del proyecto desde el punto de vista del usuario: la amigabilidad.

En primer lugar, y con el fin de alcanzar este requisito, se realizaron estudios basándose en lo siguiente:

- El concepto o idea que el usuario final tiene sobre las aplicaciones gráficas.
- El perfil de los usuarios finales que desean utilizar el software.
- La experiencia del programador en el desarrollo de aplicaciones amigables.
- El conocimiento del programador después de analizar un número importante de aplicaciones amigables ya existentes.

Con estos estudios se obtuvieron los requisitos de diseño que definieron gráficamente a la interfaz. Son éstos:

- La interfaz debe aparentar simplicidad y legibilidad a primera vista.
- La interfaz tiene que ser inteligente y guiar al usuario cuando sea posible.
- Las etapas del modelo SOLWEIG se ejecutarán a través de botones, no aparecerán dentro de los menús. Así el usuario usa lo que ve desde el principio.
- Habrá una ventana principal que muestra todas las etapas del modelo en forma de botones.
- Dado que el modelo SOLWEIG ofrece dos posibles tipos de simulación, la ventana principal será diseñada de tal manera que estas dos posibilidades queden perfectamente diferenciadas. Además, el usuario podrá ver toda la funcionalidad del sistema con este diseño.
- Cada botón representará una etapa en la ventana principal. Haciendo clic en ellos se mostrará al usuario una nueva ventana que gestionará todas las tareas y datos relativos a la correspondiente etapa.

- En la ventana principal se mostrarán dos tablas que ayudarán al usuario a conocer la situación actual de cada una de las etapas (es decir, tener constancia de qué datos se han cargado y cuáles no), librándole así el tener que abrir cada ventana de cada etapa para averiguarlo.
- Algunos botones se activarán y funcionarán en un determinado momento del proceso. Así se guiará al usuario mostrando la siguiente tarea a ejecutar.
- El usuario rara vez tecleará para introducir la entrada de datos. En su lugar, la interfaz proporcionará los componentes gráficos correspondientes para fomentar el uso del ratón.

Comentarios de los usuarios

Una vez creada la interfaz gráfica, se procedió a la realización de una serie de pruebas que demostrasen si ésta cumplía los requisitos que se han ido comentando a lo largo del resumen. Para ello se tomó un grupo de diecisiete usuarios que probaron la interfaz mediante la realización de un conjunto de tareas que se crearon con el fin de comprobar y validar el sistema. Existen dos tipos de tareas:

- **Tareas orientadas a la interfaz:** son las tareas que definen una serie de pasos a seguir sin importar la interpretación de los resultados. Tratan de demostrar si la aplicación puede ser usada incluso por aquellos usuarios que no están familiarizados con el modelo SOLWEIG o la rama de estudio.
- **Tareas orientadas al modelo:** son las tareas que enuncian una pregunta que debe ser respondida por el usuario. La interpretación de los resultados es necesaria. Tratan de demostrar si la interfaz es útil para aquellos que podrían hacer uso de ella en el futuro. Estas tareas están dirigidas a los usuarios que estén familiarizados con el modelo SOLWEIG o pertenezcan a la rama de estudio.

Los diecisiete usuarios realizaron las tareas que se les asignaron y después contestaron a un cuestionario en el que se les preguntaban sobre las distintas facetas de la aplicación (con el fin de comprobar si ésta cumplía los requisitos principales), incluyendo una opinión personal acerca de la impresión que se llevaron tras probar el software por primera vez. Tras recibir los cuestionarios y analizarlos detenidamente, se concluyó que se obtuvieron unos resultados satisfactorios que indican que la interfaz gráfica cumple sus requisitos y que, por lo tanto, será de bastante uso y satisfacción por parte del usuario final en el futuro.

Conclusiones

En este resumen, una interfaz gráfica y amigable para el modelo SOLWEIG ha sido presentada. El software soluciona el problema de compartir el modelo con otros usuarios mediante la eliminación de las limitaciones que dicho modelo manifiesta. La herramienta está escrita en Java y utiliza la biblioteca Swing para mostrar los componentes gráficos.

En cuanto a la vinculación con el modelo SOLWEIG (que está escrito en MATLAB), se han utilizado las herramientas de MATLAB Compiler y MATLAB Builder JA (de The MathWorks) para obtener una versión del modelo en Java, de manera que la interfaz gráfica puede ejecutar las funciones de MATLAB haciendo uso de un compilador en tiempo de ejecución llamado MATLAB Compiler Runtime. Este compilador se puede distribuir libremente junto con la interfaz gráfica. De esta manera, la interfaz gráfica aprovecha el potente procesado de matrices e imágenes de MATLAB para obtener los resultados de la forma más óptima; mientras que en lo relativo a su distribución, puede ser compartido con investigadores, arquitectos y planificadores urbanos sin la necesidad de adquirir licencia alguna.

Con el fin de desarrollar la aplicación, se ha seguido una metodología de trabajo, de entre cuyos métodos cabe destacar el relativo a las reuniones con los investigadores del Grupo de Climatología Urbano, a partir del cual se obtuvieron los requisitos y objetivos de la interfaz gráfica. Los requisitos han sido: interfaz amigable, extensibilidad del código y usabilidad de toda la aplicación; los cuales se han cubierto mediante el desarrollo y análisis de documentos técnicos que han definido la estructura (tanto exterior y como interior) de la interfaz. Destacar que se ha aplicado una arquitectura Modelo-Vista-Controlador, dado que cubre el requisito de extensibilidad y facilita el cumplimiento de los otros dos.

Por último, la interfaz gráfica ha sido probada por un grupo de diecisiete usuarios que han dado una primera idea del impacto del software a usuarios con y sin conocimiento acerca del modelo SOLWEIG. Los resultados pueden considerarse como satisfactorios, ya que la mayoría de las preguntas contestadas por los usuarios han tenido una respuesta positiva.

Abstract

SOLWEIG is a computer software model which can be used to estimate and analyse the complex interaction between urban design and the thermal environment. We have extended this work by building a computer software and graphical user-friendly tool that integrates the SOLWEIG model.

The tool is written in Java, while the SOLWEIG model is in MATLAB. It has been tested and evaluated by both normal users and researchers who are familiar to the SOLWEIG model. This tool is a crucial step to allow sharing the model to the users, such as architects and urban planners.

Table of contents

List of figures

List of tables

1. Introduction

The aim of this thesis is to introduce and explain in detail all the information related to the "SOLWEIG – a climate design tool" project, a graphical and user-friendly software application that integrates and executes the software model called SOLWEIG. SOLWEIG is a radiation model that can make climate estimations (such as sunshine durations, shadow patterns and daily shading) and analyse the complex interaction between urban design and the thermal environment. It was developed by the Urban Climate Group of the Department of Earth Sciences, University of Gothenburg, Sweden.

The SOLWEIG model has been proved to be a useful tool for those who are interested in the subject. However, some restrictions, like being a command-line based software or the need of buying a license to execute it, makes it difficult to share with other users. This is the reason that encouraged the researchers from the Urban Climate Group to ask for a graphical and user-friendly tool that would link the SOLWEIG model and solve these sharing limitations.

For the development of the tool, three requirements have been taken as targets: user-friendly, extensibility and usability. While the extensibility is targeted to the future developers of the tool, in the sense that the software will be easy to extend and update, the user-friendly and usability ones have the aim of ensuring a good looking, elegant and easy to use interface that will encourage the users to use it more than one time. In order to measure the fulfilment of these three requirements, the tool has been tested and evaluated by both normal users and researchers who are familiar to the SOLWEIG model.

The graphical interface has been written in Java and integrates the SOLWEIG model, which is written in MATLAB. Thus, one of the main advantages of this tool is that it offers an alternative way to use a MATLAB application by combining it with a powerful graphical interface.

It is worth to mention that this tool is useful for those who are researching in the subject, as well as for architects and urban planners. The tool is free and executes the SOLWEIG model by using the MATLAB Compiler Runtime, which can be distributed royalty free along with the graphical interface.

1.1. Overview of the document

The thesis contains the following sections:

- **The SOLWEIG model:** it introduces and describes the SOLWEIG model and its applications in real life.
- **Objectives:** it reasons the need of developing a graphical user-friendly interface for the model and explains the methodology followed to achieve this goal.
- **The graphical user-friendly interface for SOLWEIG:** it explains in detail all the steps taken to obtain the final graphical user-friendly interface for the model. These steps can be gathered in three groups: the programming language, the integration of the model within the interface and the developing of a user-friendly interface.
- **Feedback of the testers:** it shows the opinions and feedback obtained by a group of testers that have already tested the software.
- **Conclusions:** it summarises the contents of the whole document by pointing out briefly the main information of it.

2. The SOLWEIG model

This section introduces and describes the SOLWEIG model, including its inputs and outputs. Besides, the applications of this model are listed at the end of the section.

2.1. What is SOLWEIG

SOLWEIG comes from the acronym SOlar and LongWave Environmental Irradiance Geometry, and it is a software solution that estimates spatial variations of three-dimensional radiation fluxes and mean radiant temperature (T_{mrt}) in complex urban settings. It was developed by the Urban Climate Group of the Department of Earth Sciences, University of Gothenburg, Sweden.

The T_{mrt} is one of the most important meteorological factors that govern human energy balance and the thermal comfort of man outdoors. It can be defined as:

"The sum of all short and long wave radiation fluxes (both direct and reflected), to which the human body is exposed." [3]

The SOLWEIG model bases its origins from a sustainable urban design perspective, with the aim of determining how the radiation fluxes and the T_{mrt} can affect the health and wellbeing of the humans in a concrete urban setting along a concrete day.

The model is targeted to the researchers of the subject, architects and urban planners.

2.2. Inputs and outputs

SOLWEIG bases its calculations on a map (see Figure 1), which represents spatial variations of urban geometry, called DEM (Digital Elevation Model). The T_{mrt} is derived from this data plus some other inputs, "such as direct, diffuse and global shortwave radiation, air temperature, relative humidity and geographical information (latitude, longitude and elevation)" [3].

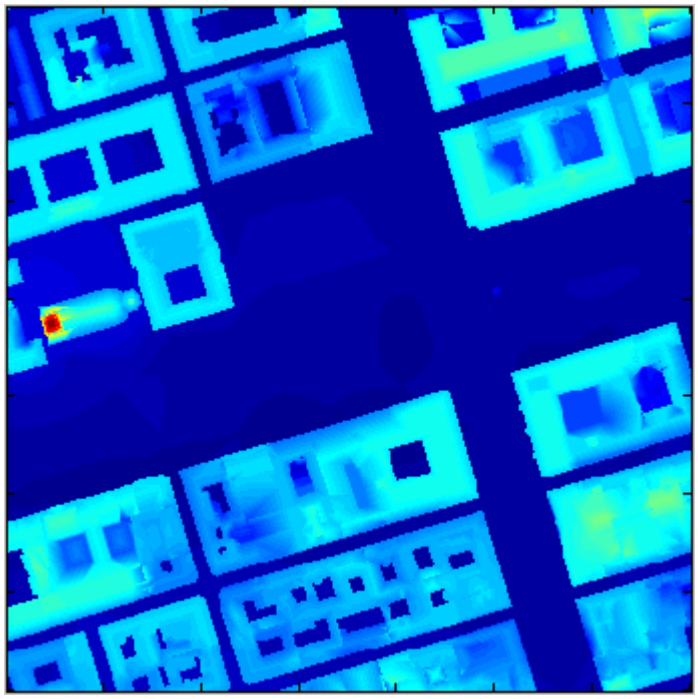

Figure 1: DEM example

In order to show the results, the model generates three different types of outputs:

- **Maps:** a set of twenty-four maps that show the spatial variations of the T_{mrt} for each hour of a concrete day plus one more that contains the average of all this information (see Figure 2).
- **Diagrams:** three graphical diagrams that show the spatial variations of the T_{mrt} and both the short and long wave radiation fluxes in a concrete point of the DEM (see Figure 2).
- **Data files:** a text version of the previous maps and diagrams which are stored in the user's computer.

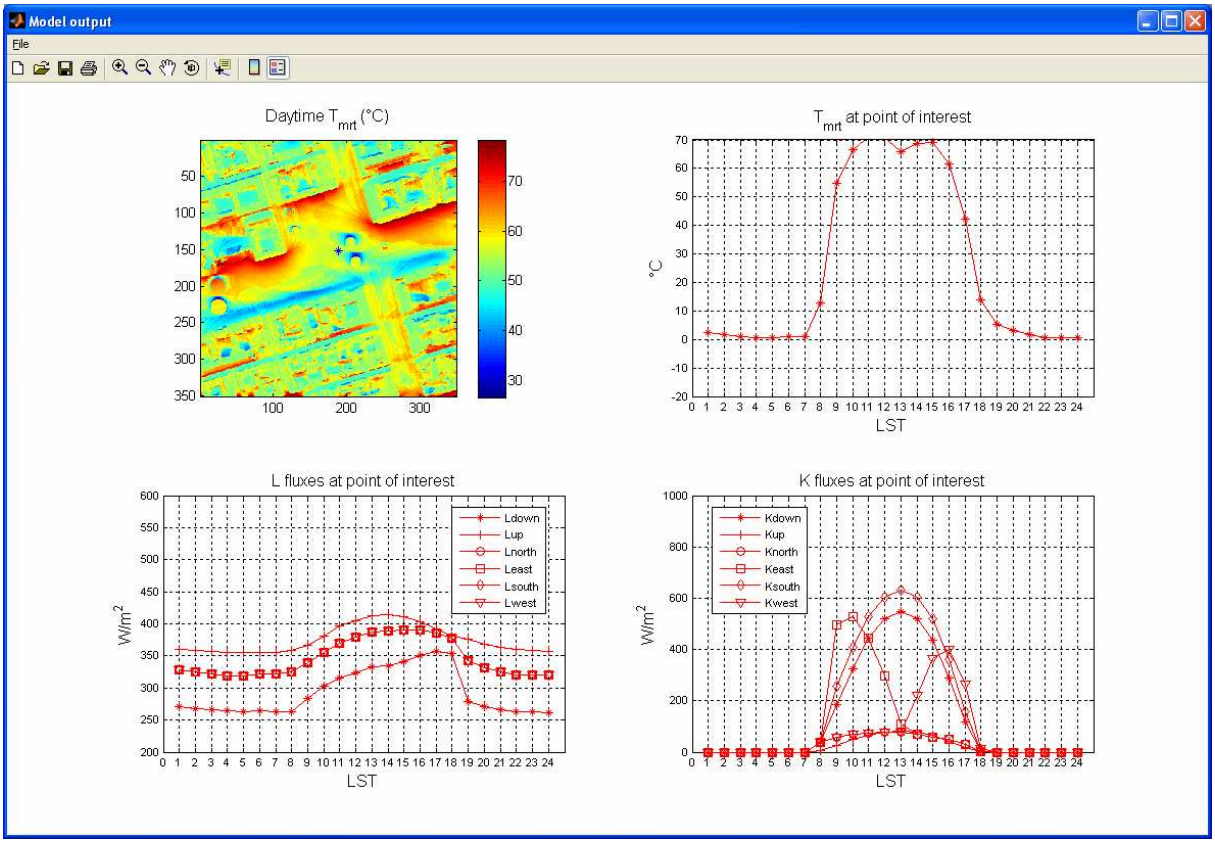

Figure 2: SOLWEIG's output example

Figure 2 is an example of the type of maps and diagrams that the SOLWEIG model produces as outputs. The map at the top-left side of the figure corresponds with a map with the averages of the T_{mrt} on the urban setting that the DEM represents (note that the vertical bar at the right side of the map measures the T_{mrt} from 30ºC to more than 70ºC). Then the three diagrams show the evolution of the T_{mrt} (top-right side) and long (bottom-left side) and short (bottom-right side) wave radiation fluxes along the day in a concrete point of the DEM.

SOLWEIG is written in MATLAB programming language [5]. This involves a certain number of advantages for the aim of this model, as matrices processing are required continuously, requirement that MATLAB covers perfectly. Therefore better, fast and efficient results are obtained.

2.3. Applications of SOLWEIG

The SOLWEIG model can be applied for the following targets:

- Climate estimations (shadow pattern, radiation fluxes, mean radiant temperature, etc) in urban settings.
- Analyse the interaction between urban design and thermal environment.
- Creation of thermal comfort maps.
- Analyse the local effect of a changing climate in the urban setting.

3. Objectives

This section gives an argument for the need of developing a graphical user-friendly interface for the SOLWEIG model. Once this becomes a fact, the methodology that has been followed to achieve this goal is provided.

3.1. Need to develop a graphical user-friendly interface

Once the SOLWEIG model was proved to be useful, the need of sharing it with some groups of researchers and other persons that might be interested in using it arose. However, some limitations were found regarding the usability, quality and performance of the software, such us:

- The SOLWEIG model is using a command-line as interface. This means that, in order to communicate with the software, the user has to type commands and the input data in the command-line to perform the corresponding tasks.
- Neither windows nor buttons are provided during the simulation, objects that nowadays are typical and very common to be used from the user side when he/she deals with software applications.
- The model is working in such a way that it asks the user for the next input data to be typed. It follows a sequential and sorted flow, split in steps, before getting the results and ending the execution. In this way the user is forced to restart (and thus retype) the model every time he/she wants to execute the software and obtain some new results.
- Any mistakes when typing or loading a file forces the user to restart the whole simulation again.

These limitations show that the model is not user-friendly, as it becomes difficult to reuse it many times in a comfortable and friendly way.

In order to share the model to future users, it is needed to provide at least a user-friendly interface that might ensure usability, a good performance and, if possible, an acceptable quality of service. These requirements can be fulfilled by developing a graphical user interface, which is able to solve all the limitations that a command-line interface has. Concretely in this case, with a graphical user interface the user can:

- Make use of windows and buttons, instead of typing commands and the input data, in order to execute the different tasks.
- Break the sequential flow and reload or reselect the input data from a previous step.
- The software does not end after getting the results; therefore the user can re-execute it as many times as needed without restarting the application and reloading the data.
- Any mistakes when loading or selecting the data will not force the user to restart the application.

These advantages encourage the idea of developing both a graphical and a user-friendly interface in order to achieve the main target: to ensure the productivity and constant use of the software from the user side by providing it in an easy and visual way.

3.2. Methodology

Once the need of developing a graphical user-friendly interface becomes a fact, a methodology that achieves this goal has been followed. In order to explain the different methods that have been applied in this project, first it is necessary to group the main tasks that have involved the process of developing the software:

7) Understand the SOLWEIG model and the theory behind it.
8) Analyse the source code of the SOLWEIG model to extract the requirements, software functionalities and formats.
9) Update the source code of the SOLWEIG model in such a way that it can be integrated within the graphical interface.
10) Decide the programming language for the graphical interface by taking into account some important requirements: code integration, user-friendly, usability, extensibility (for future versions of the model) and quality of results.
11) Develop the graphical interface by paying special attention on the user-friendly, from the user point of view, and the extensibility requirements, from the programmer point of view.
12) Test whether the previous requirements are fulfilled.

In order to solve these tasks, different methods have been applied efficiently. The following list describes them:

f) Meetings with the researchers from the Urban Climate Group at the Department of Earth Sciences (University of Gothenburg, Sweden) to solve tasks 1 and 4.
g) Meeting with the software developer of the SOLWEIG model (from the Urban Climate Group at the Department of Earth Sciences, University of Gothenburg, Sweden) to discuss task 2 and solve task 3. It is worth to mention that the development of use cases has been the way that both the model and graphical interface developers have been able to communicate and understand the problem in a very graphic and easy way.
h) Develop the user and software requirement specification (URS and SRS) documents to discuss tasks 4 and 5 and solve task 2.
i) Use the corresponding programming platforms to solve task 5.
j) Collect feedback from a group of software testers to solve task 6.

Besides, weekly meetings and presentations to the supervisors of the project have been held with the aim of keeping track of the progress of the work done during the week before.

4. The graphical user-friendly interface for SOLWEIG

This section explains in detail all the steps that have been taken to obtain the graphical user-friendly interface for the SOLWEIG model.

First, an argument about the chosen programming language for the graphical interface is provided as well as a discussion that compares it with some other programming language alternatives. Second, the way the SOLWEIG model is integrated within the graphical interface is explained. Third, a set of technical information regarding the user needs and system features is provided by setting the user and system requirements, the architecture and the design. Finally, the appearance of the system is shown by means of screenshots that measure the user-friendliness of the system.

4.1. The programming language

The first point regarding the graphical interface was to make a decision about the programming language to be used. As the SOLWEIG model is developed in MATLAB [5], the initial idea was to try to adapt the command-line interface that the model was based on to a graphical one. MATLAB offers a set of toolboxes that allow the software developer to create windows-based applications, so that it could have been possible to rewrite the code and adapt it to a new version by using the MATLAB programming language by itself. However, there are some limitations that strongly encourage leaving this option. These limitations are the ones listed below:

- MATLAB is a proprietary product of The MathWorks [9], where users are dependent on a vendor for products and services (this case on MathWorks). This means that, in order to execute the model, the end user must have a valid license of MATLAB installed in his/her computer. This license is not freeware [10] and might provoke the negligence of the user of buying one and, therefore, of using the software.
- The fact of being sensible that actually there are more powerful graphical libraries that belong to other programming languages that would offer higher amount of options and possibilities to the developers to achieve one of the main targets of the project: a user-friendly interface.

Once having rejected this option and due to a MATLAB limitation when exporting the code to a certain programming language (see section 4.3), the selected language was Java [11]. Among all the features this language has, it is worth to mention the following:

- It is one of the most known and used programming languages to develop graphical and wed-based interfaces. It offers a very powerful graphical library called Swing [12] that contains the necessary features to get a very user-friendly interface.
- It is freeware and only requires a virtual machine installed in the computer of the end user in order to be able to execute any Java applications.

With these advantages it is possible to achieve the main goals of the project, so that this programming language has been the one used for the development.

Regarding the programming platform, NetBeans IDE 6.5 [17] has been used to develop the whole graphical interface, which is written under Java version 6.

4.2. Other programming languages

Besides the Java programming language, there are another two alternatives that could have been chosen as the graphical interface programming language. These two languages are:

- .NET [13].
- C++ [14].

In order to understand why there are not more alternatives, please see section 4.3.

Both alternatives are object oriented [15] and could have been perfectly used for the aim of the project. However, there were some constraints that finally encouraged making the decision of using the Java programming language. They are listed below:

- If we compare this with the other two alternatives (Java and .NET), the C++ standard library is not as rich as the other two ones, especially when looking for components to develop a graphical user interface [18].

- .NET is a language that can be perfectly compared with Java. In fact nowadays both are two of the most used ones for developing graphical-based interfaces. Even the grammar and syntax of both languages have certain similarities. In this case .NET was rejected because potentially the same results would have been obtained with Java and especially because the developer was more familiar with the Java programming language.

4.3. Linking the SOLWEIG model with the interface

After making the decision of not using the MATLAB programming language to develop the graphical interface, the need of linking the SOLWEIG model (written in MATLAB) with another programming language (finally Java) arose. The first idea was to rewrite the code of the model into Java but, due to certain and important drawbacks, like not having the powerful processing of matrices and plots of MATLAB, this option was rejected.

As MATLAB is a proprietary product, it was mandatory to check the different solutions the vendor offered for this purpose. It was found out that a new set of tools called MATLAB Compiler and MATLAB Builder, under the Application Deployment tools section, was provided to deploy MATLAB functions as library files which could be used with C, C++, Excel, .NET or Java application building environment. That could be the solution to link the model with one of the proposed programming languages, from which the following were suitable for the target of the project, as it had to be object and graphical oriented:

- C++.
- .NET.
- Java.

As it has been explained before, the selected programming language was Java. Therefore, the corresponding MATLAB component that has been used is MATLAB Builder JA [16], which gathers the MATLAB functions of the SOLWEIG model in one Java library file.

The drawback of this solution is that the computer where the application has to be deployed needs to use a runtime engine called the MCR (MATLAB Compiler Runtime) for the MATLAB files to function normally. This means that, besides the graphical interface, an extra application has to be installed; thus it would take more time to the user and more computer resources. However, the MCR is provided with MATLAB Compiler and can be distributed freely with the library files generated by the MATLAB Compiler.

One of the main advantages of this solution is that the user has an alternative way to use MATLAB applications along with a powerful graphical library without buying any licenses and additional software.

4.4. User requirements

One of the first tasks to be studied and done was to extract the different main steps, software functionalities and formats of the SOLWEIG model in its command-line version. This work was understood as a way of realizing the user needs when using the model and, therefore, the future graphical interface. This section provides a list of all the functional requirements of the system that come from the user point of view. They are the ones the graphical interface is based on and are shown as use cases in Figure 3:

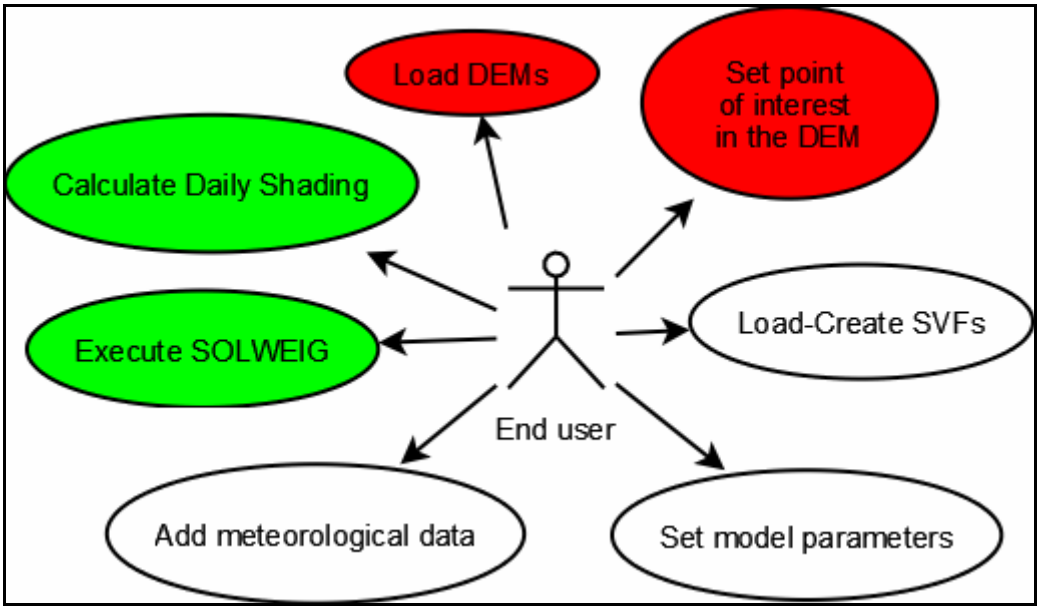

Figure 3: Use case diagram

Some use cases have some important issues in common; therefore they have been equally coloured. The meaning of the different colours is explained below:

- **Green use cases:** “Execute SOLWEIG” and “Calculate Daily Shading”. They correspond with the last step from one of the possible flows the application offers. They execute a part of the model and show some results.
- **Red use cases:** “Load DEMs” and “Set point of interest in the DEM”. They are common functionalities for all the final steps (the green use cases).
- **White use cases:** “Load-Create SVFs”, “Set model parameters” and “Add meteorological data”. In this case they are steps for the “Execute SOLWEIG” use case.

4.5. System requirements

Basing on the user needs and having them gathered in the corresponding use cases (or steps), the next point was to develop each step by describing the way the graphical interface was going to react after the interaction of the user with the system, defining what are called the software (or system) requirements.

The software requirements have been developed by taking into account one of the targets of the project: usability. Concretely, the following features have been applied:

- The buttons are handled in such a way that they guide the user during the flow in most of the cases.
- The names of the windows and messages of the dialogs have been carefully selected in order to be self-explanatory and conclusive.
- There is no way to reach a fourth-level window by starting from the main frame, so that the user can keep track on a task that is being performed without forgetting or not understanding its goal.

For a complete and detailed explanation of all the software requirements, please see section 7.

4.6. Architecture of the system

The functionality of the system is obtained with the software requirements, explained in section 4.5, but it does not give a clear idea about the organization of the data and the way it is handled internally. In order to explain this, the architecture of the system is going to be shown and described in this section.

The architecture of the system has been based in the most important target of the project from the programmer point of view: the extensibility of the code. It is more than probable that the model will be updated quite often and that it will be added more and new functionalities along these changes. Therefore the graphical interface must be prepared to easily be adapted to these future updates without involving big and crucial changes on it.

It is important to isolate the code of the model from the code of the graphical interface. As it was explained in section 4.3, the code of the model will have its version in Java as a Java library. Thus, the architecture of the system must separate the graphical components from the references to the Java library that represents the SOLWEIG model.

Based on the previous constraint and taking into account the extensibility requirement, a Model-View-Controller architecture was chosen to organize the internal flow of the system. This architecture can be seen in Figure 4:

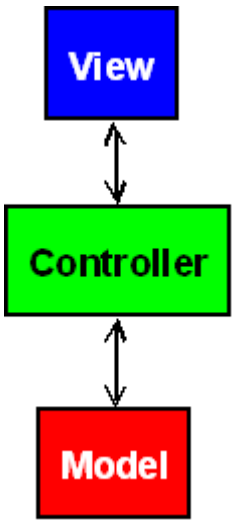

Figure 4: Model-View-Controller architecture

Where each component has the following role:

- **Model:** contains and handles the references to the SOLWEIG model. It gets the inputs that come from the Controller, calls the different MATLAB functions, saves the output data and return the results to the Controller again. Internally it is organized in such a way that each step (or use case) is a sub-module of the module, so that removing or adding new steps will not affect the others (extensibility requirement).
- **View:** contains and handles all the windows and dialogs that form the graphical interface. It gets the inputs that the user must load or select, send them to the Controller and displays the results that come back from it to the user again. Internally it is organized in such a way that each step (or use case) is a sub-module (dialog) of the module (main frame), so that removing or adding new steps will not affect the others (extensibility requirement).
- **Controller:** it is the module that controls the whole internal flow of the system. It is the link between the View and the Model components. It gets the inputs from the View, sends them to the Model, gets results from the Model and returns them to the View.

With this architecture, the data that belongs to the SOLWEIG model is completely separated from the one that forms the graphical interface. Besides, the different updates of the SOLWEIG model may affect the interface in different ways, but it will not mean big changes on it. The possible cases are listed below:

- An update in the model that does not change any inputs and outputs in the corresponding function will not involve any changes in the interface, neither in the Model component.
- An update in a function that changes some inputs and/or outputs will involve to update the Model, the Controller and rarely the View, but in a small scale.
- Adding a new function that involves the creation of a new step will force to update all the components. Updating the Model and the Controller will not entail big changes (just adding the new sub-module in the Model and a reference to it in the Controller); while the View will depend on the design of the interface regarding the flow in which this new function must be inserted (see section 4.7).

4.7. Design of the system

Once the architecture of the system has been defined, the next step is to design all the different components that have been created during the previous process. Concretely, the three components to be designed have been:

- Model.
- View.
- Controller.

As it was explained in section 4.6, both the Model and the View would contain the main data of the whole application, while the Controller would be in charge of linking them two. Therefore it has been needed to find a design that could facilitate the connection between the two data components in a easy way. Due to the extensibility requirement, the most pertinent idea has been to subdivide the corresponding data component in sub-modules where each one of them would represent a step (or use case) coming from the user requirements. Thus, by taking the step “load DEMs” as an example, the different components would have the following:

- **Model:** a sub-module called “load DEMs data” that would handle all the MATLAB data and functions related to this step.
- **View:** a sub-module (dialog) called “load DEMs dialog” that would allow the user to loading or selecting all the inputs and to getting all the outputs related to this step.
- **Controller:** a reference to the sub-module of the Model and to all its functions in order to get the results and send them to the View.

Regarding the View component, besides its internal design, it has been also necessary to set the graphical requirements in order to get appearance of the system, which has been defined basing on the most important target of the project from the user point of view: the user-friendly requirement. Besides the two already-set requirements, usability and extensibility, the user-friendly is the last one that would make the graphical interface fulfilled its main goals.

In order to achieve this requirement, some studies were taken based on the following:

- The current idea that a user has when defining and using a graphical application.
- The profile of the end-users that will like to use the software.
- The previous experience of the programmer when developing user-friendly applications.
- The knowledge of the programmer after analysing an important number of user-friendly applications.

Finally, the main ideas that would define the requirements of the graphical interface were obtained. They are the following:

- It has to look like simple and easy to understand as a first sight.
- It has to be smart and guide the user when possible.
- There will not be tasks inside menus, just within buttons. The user uses what he/she sees.
- There will be a main frame that will only contain the steps (or use cases) in the shape of buttons.
- As there are two possible executions, the main frame will be designed in such a way that these two flows will be perfectly differentiated. Besides, the user will be able to see all the functionality of the system with this design.

- Each button will represent a step. Clicking on it would show the user a new dialog that will manage all the tasks and data related to that step.
- In the main frame two status tables will help the user to know the current status of each step without forcing him/her to know that by opening all the dialogs every time.
- Some buttons will be enabled and will work only when they have to (smart software and guiding the user).
- The user will rarely type any input data. Instead, the interface will provide the corresponding components that will encourage the user to clicking.

All these requirements can easily be seen in section 4.8.

4.8. Appearance of the system

Once the technical information about the system has been set, the developer is ready to start the development of the interface by following all the requirements and constraints that have been previously defined. This section describes the appearance of the system by showing the screenshots of the final application. First the main frame, with its different appearances, is presented as the main window of the interface. Then the dialogs that represent the different steps of the model are displayed. Note that this section covers the software from the graphical point of view; the functionality of each window, representing the contents of the SOLWEIG model, is not explained.

4.8.1. Main frame

Figure 5 shows the initial window (or main frame) that the user will find every time he/she launches the application:

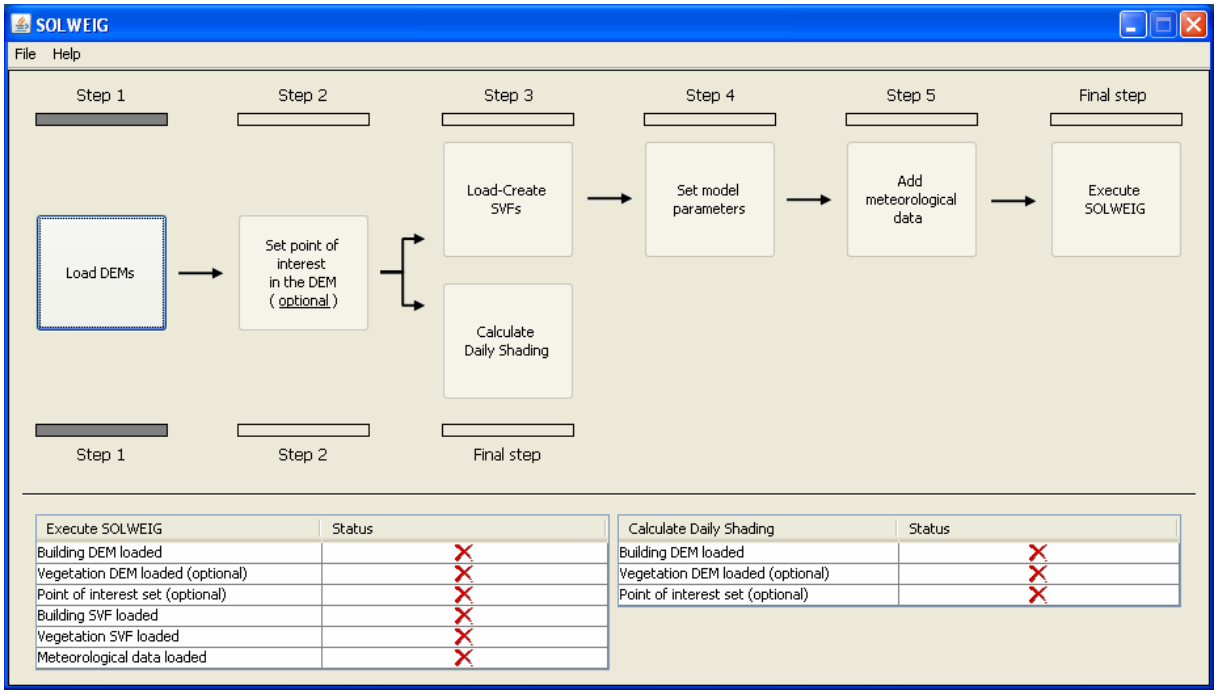

Figure 5: Main frame at the beginning

As it can be seen in Figure 5, the different steps of the SOLWEIG model are shown from the beginning in the shape of buttons. They are sorted in the corresponding two flows (located between the menu bar and the status tables at the bottom of the frame). The first one has six steps, starting from the “Load DEMs” step and ending with the “Execute SOLWEIG” one. The second one has three steps, from “Load DEMs” to “Calculate Daily Shading”. Notice that the steps one and two (“Load DEMs” and “Set point of interest in the DEM”) are common for both flows.

Besides, the status tables (located at the bottom of the frame) indicate the current situation of the input data (that is, the files and information that must be loaded by the user in order to execute the SOLWEIG model). At this beginning point, they are all unloaded (marked by red-cross icons). There is one table per flow, so that in this case the one at the left side represents the input data from the first flow, while the one at the right is related to the second flow.

Regarding the buttons, there is only one which is allowed to be clicked, so that the user will not have to think or find out how to start using the software (although this would be known for those who are related to the subject). In this case this button represents the step one.

Figure 6 shows the contents of the menu bar that belongs to the main frame:

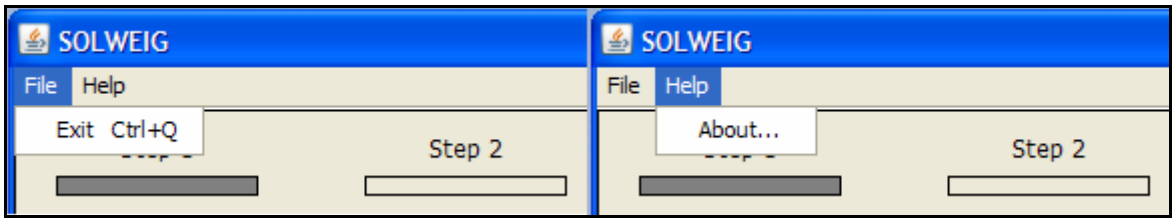

Figure 6: Main frame's menu bar

The menu bar from Figure 6 does not contain any tasks related to the SOLWEIG model; instead it shows basic functionality of the interface, like in this case are to exit the tool and an about section that would retrieve the information about the interface (description of the model, software version, developers, etc.).

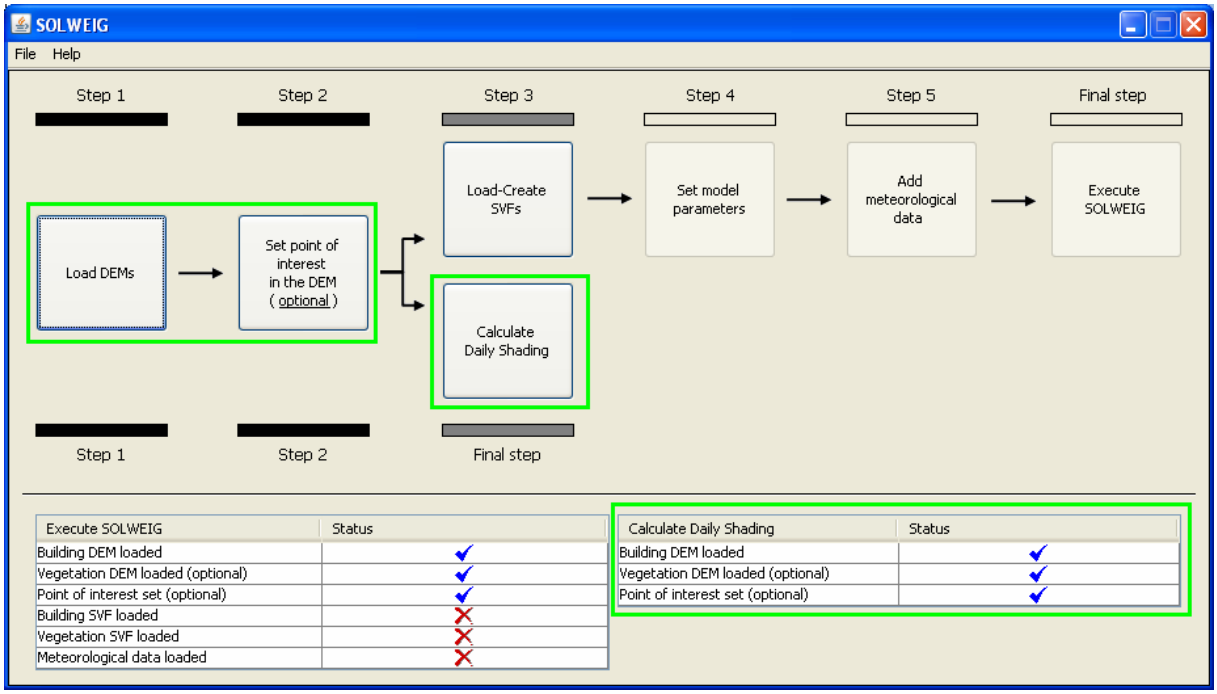

Figure 7: Main frame with the second flow ready

After loading the first required files (those corresponding to first step – “Load DEMs”), the buttons at the main frame start being able to be used and will allow the user to go further along the steps (see Figure 7). In this case the second flow (“Calculate Daily Shading”) is ready to be simulated and get the results. Besides, the corresponding status table (bottom at the right) indicates (with the blue-tick icons) that the inputs are loaded, so that it helps the user to understand the current situation of the system.

Figure 8 shows the main frame when all the input data related to the first flow is loaded as well:

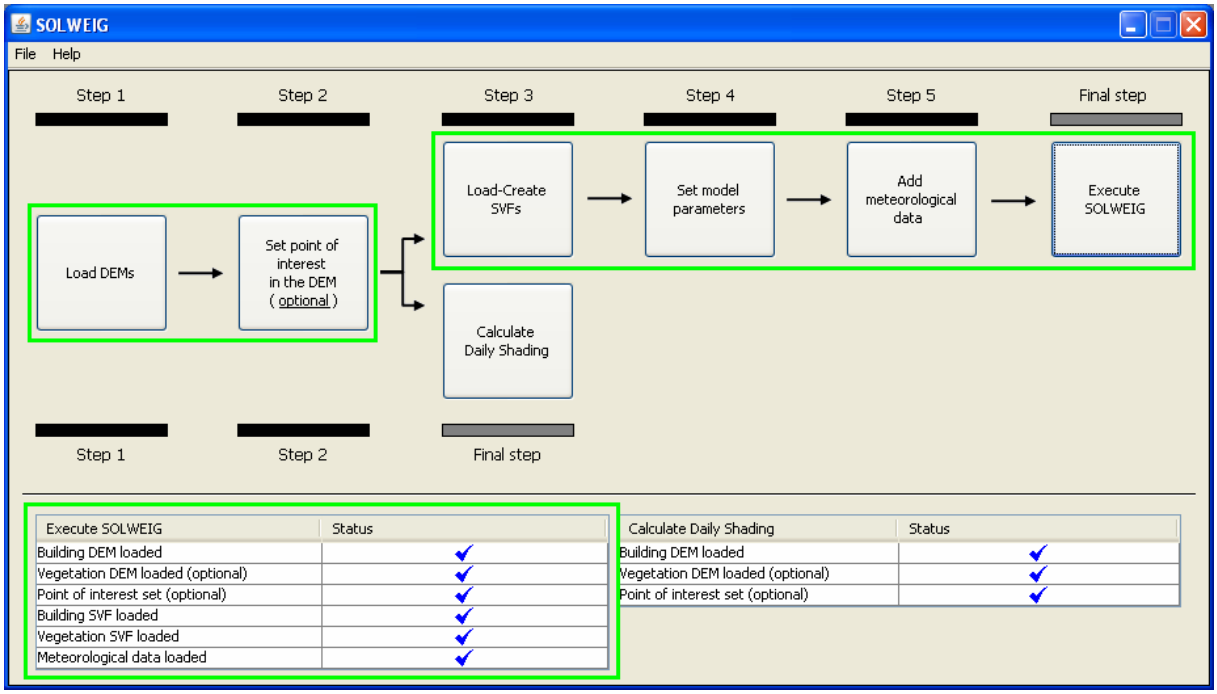

Figure 8: Main frame with the first flow ready

This is the case where all the buttons are enabled. Therefore the first flow can also be simulated. In fact, the two status tables have the blue-tick icons next to all the inputs.

4.8.2. Load DEMs

When a button from the main frame (see Figure 6) is clicked, a new dialog pops up with all the functionality and input data related to the step it represents. In this case the first step ("Load DEMs" button) pops up the dialog as shown in Figure 9:

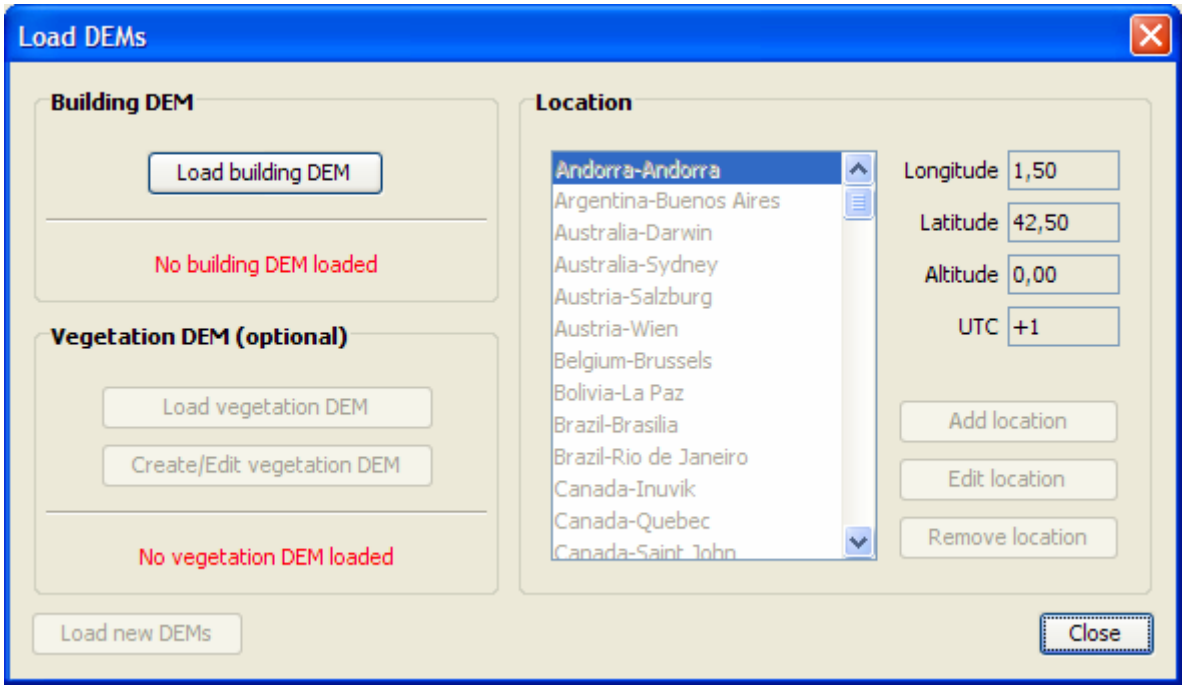

Figure 9: Load DEMs step at the beginning

Figure 9 shows how the system guides the user by specifying the action that has to be performed in order to load the input data correctly (this is done by enabling the corresponding button, in this case called "Load building DEM").

After loading the corresponding data, the system enables all the buttons shown in Figure 9. By clicking the "Create/Edit vegetation DEM" button, the user will get two new dialogs (see Figure 10 and Figure 11):

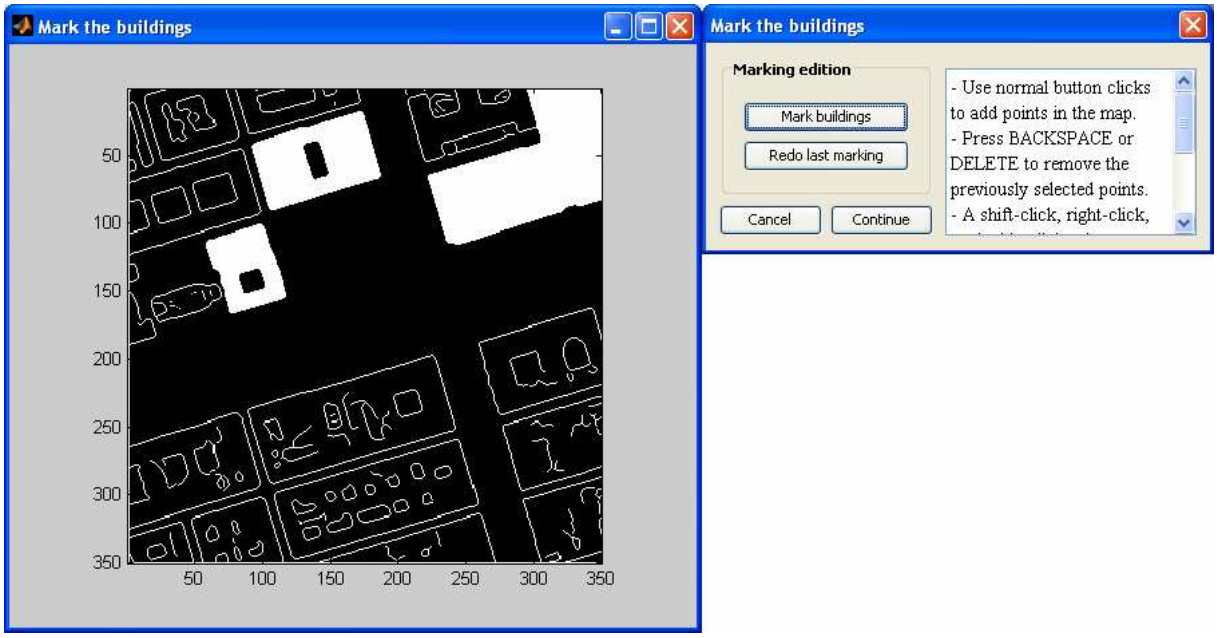

Figure 10: Load DEMs step when marking the buildings

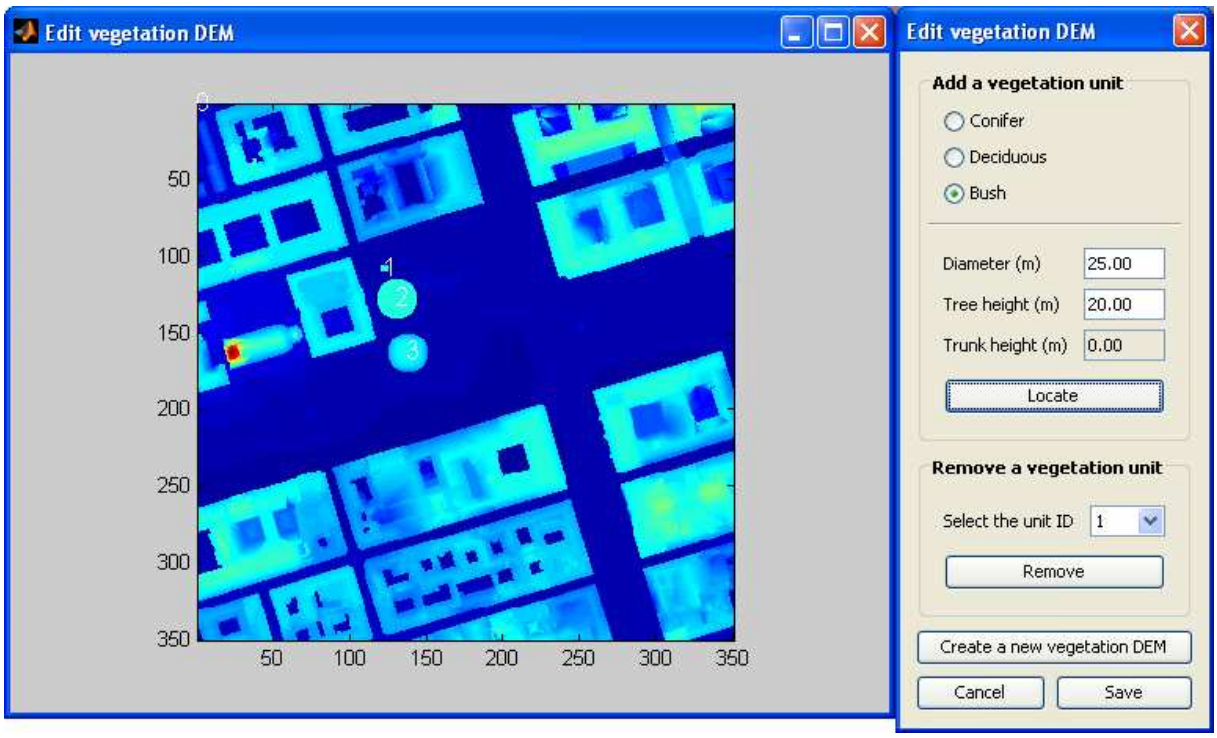

Figure 11: Load DEMs step when setting the vegetation

These two dialogs from above represent a third-level dialog, while their parent's dialog (in this case "Load DEMs") represents a second-level one. This is the furthest the user can get by clicking and getting new windows along the steps. Therefore it is easy for the user to keep track on the task that is being performed.

Figure 12 shows how the “Load DEMs” dialog looks like after all the corresponding input data has been loaded:

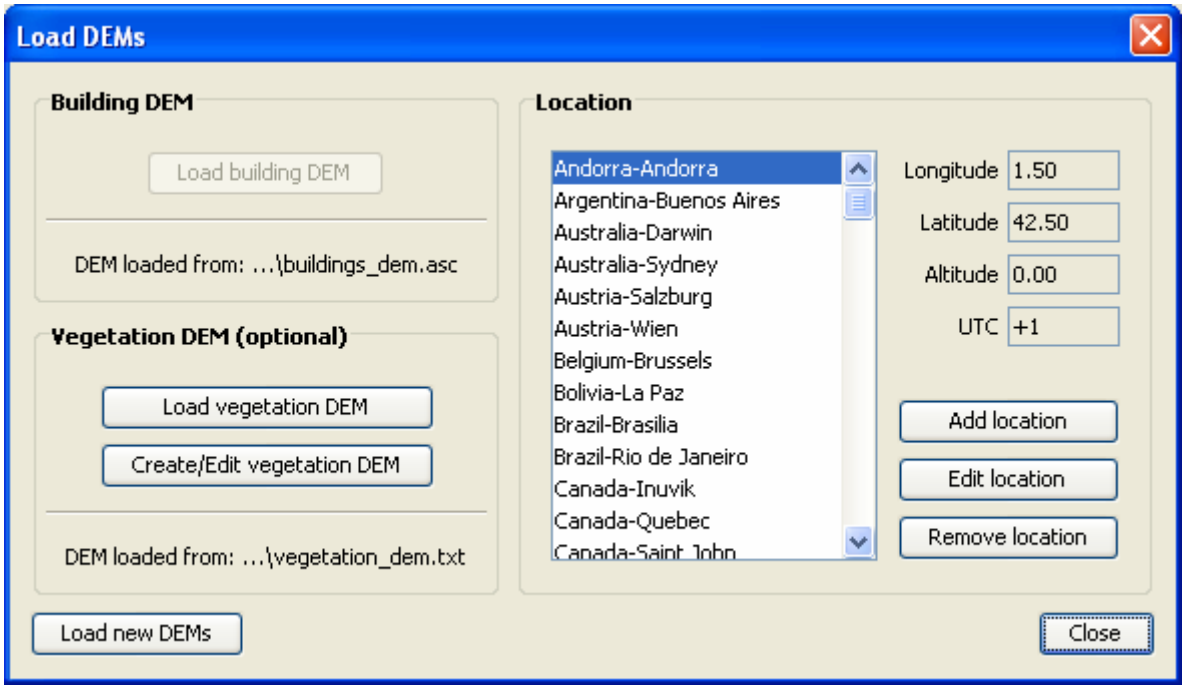

Figure 12: Load DEMs step when both DEMs are loaded

By clicking on “Close” (located at the right-bottom), the dialog will be hidden and the user will go back to the main frame, which now will get the appearance shown in Figure 7.

See section 7.1 for a detailed description of this step from the system point of view.

4.8.3. Set point of interest in the DEM

Figure 13 corresponds to the dialog that is popped up when the button “Set point of interest in the DEM” is clicked (see Figure 7):

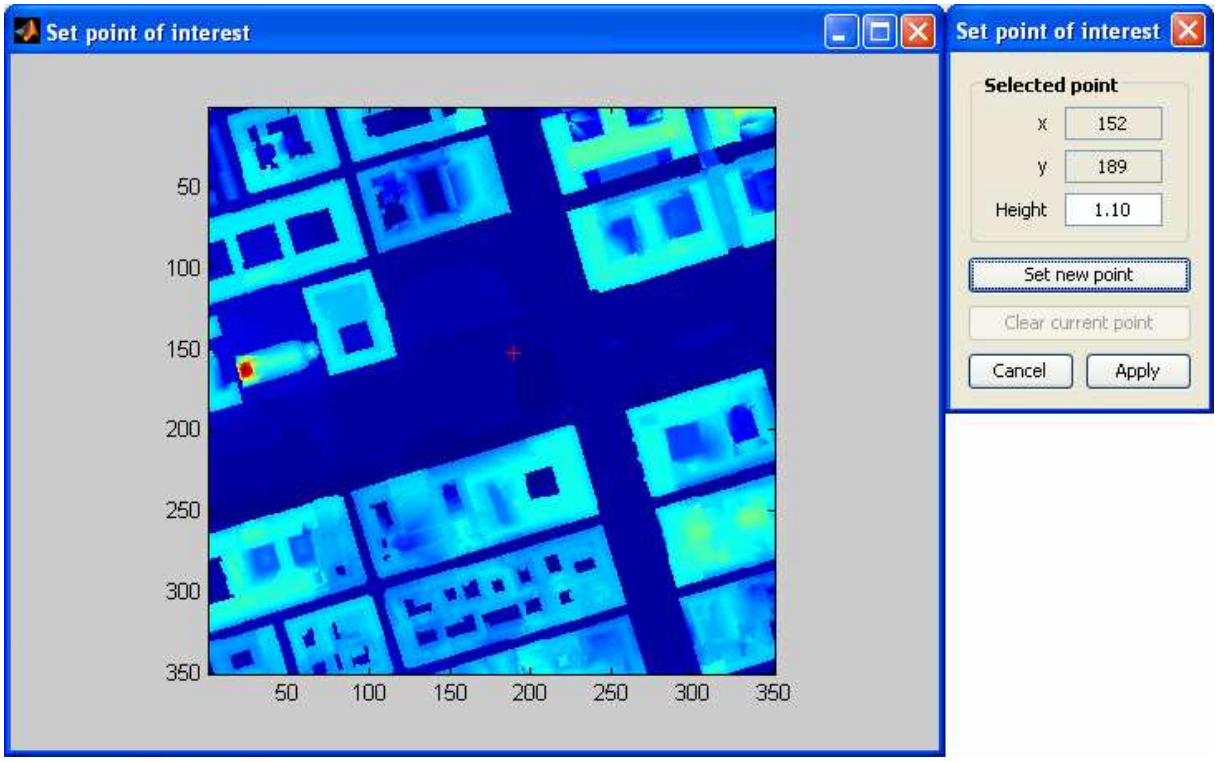

Figure 13: Set point of interest with point set

Figure 13 is an example that shows how the Java interface integrates the SOLWEIG model in MATLAB, as the left window is a MATLAB one with all its functionality, while the one at the right is a Java window that controls the MATLAB one. In this case, clicking on the “Set new point” button will activate the MATLAB window for the user to click on the shown DEM. After the interaction of the user, the Java window will get the selected coordinate and show it on its window (“x” and “y” boxes at the top of the Java window).

See section 7.2 for a detailed description of this step from the system point of view.

4.8.4. Load-Create SVFs

Figure 14 corresponds to the dialog that is popped up when the button “Load-Create SVFs” is clicked (see Figure 7):

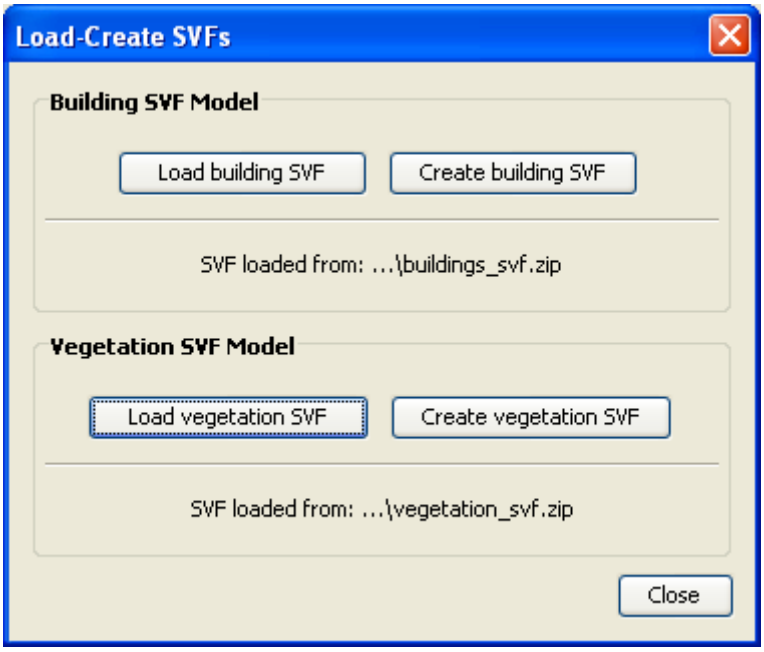

Figure 14: Load-Create SVFs when both SVFs are loaded

Like the dialog shown in Figure 14 encourages, the user will normally click instead of typing the input data. This is normally a more comfortable and efficient solution for the users.

As shown in Figure 14, the input data is already loaded, thus by clicking on “Close” (located at the right-bottom) the dialog will be hidden and the user will go back to the main frame, which now will enable the buttons corresponding to the fourth and fifth steps of the first flow.

See section 7.3 for a detailed description of this step from the system point of view.

4.8.5. Set model parameters

Figure 15 corresponds to the dialog that is popped up when the button “Set model parameters” is clicked (see Figure 7):

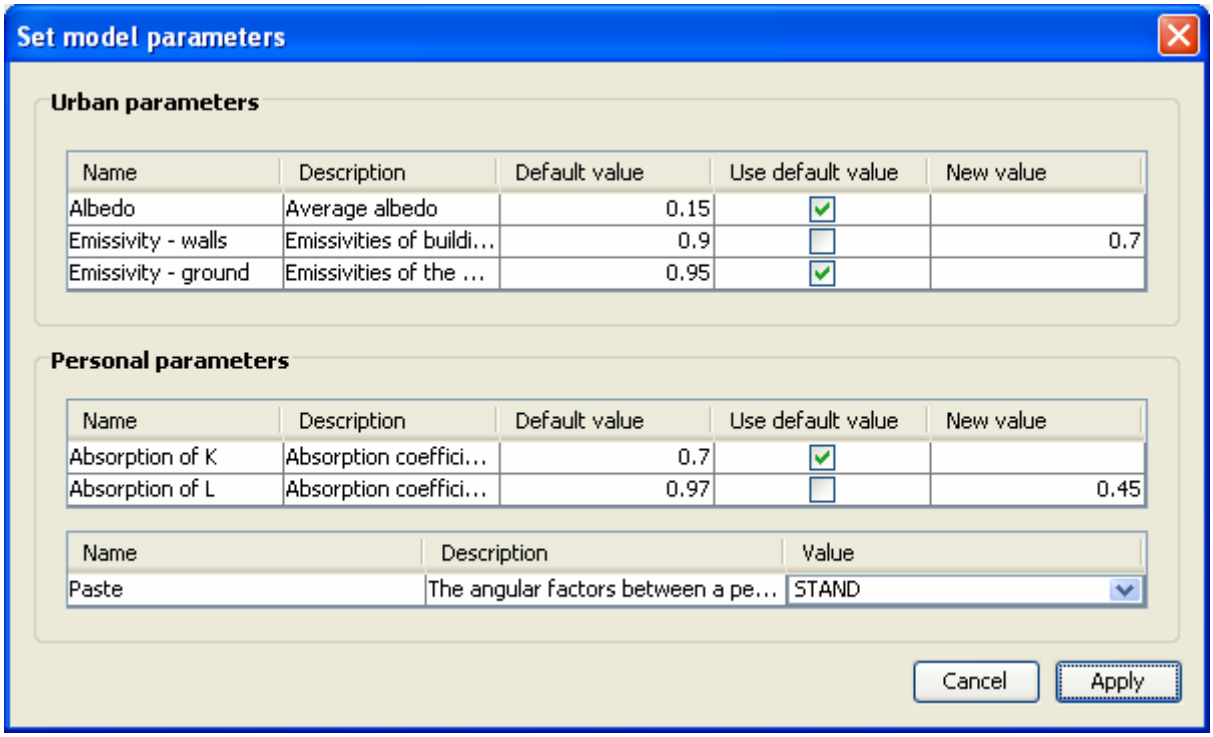

Figure 15: Set model parameters with and without default values

See section 7.4 for a detailed description of this step from the system point of view.

4.8.6. Add meteorological data

Figure 16 shows the dialog that is popped up when the button “Add meteorological data” clicked (see Figure 7):

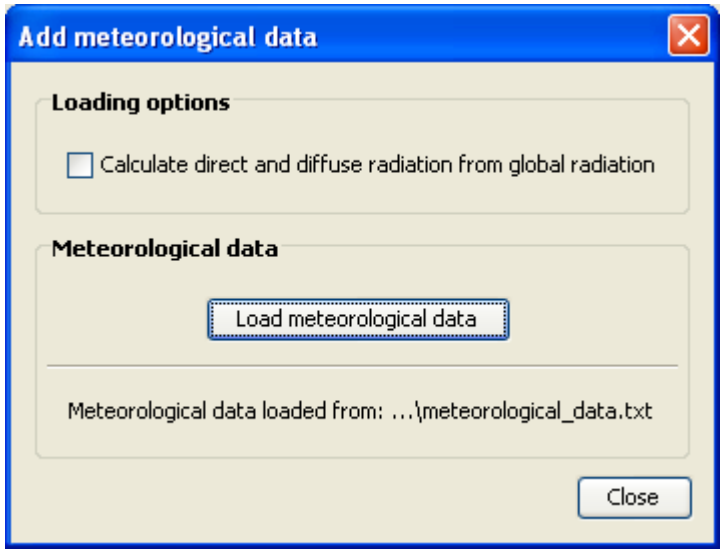

Figure 16: Add meteorological data with data loaded

In the above figure the input data is already loaded, thus by clicking on “Close” (located at the right-bottom) the dialog will be hidden and the user will go back to the main frame, which now will get the appearance shown in Figure 8.

See section 7.5 for a detailed description of this step from the system point of view.

4.8.7. Execute SOLWEIG

Figure 17 shows the execution dialog that is popped up when the button “Execute SOLWEIG” in the first flow is clicked (see Figure 8):

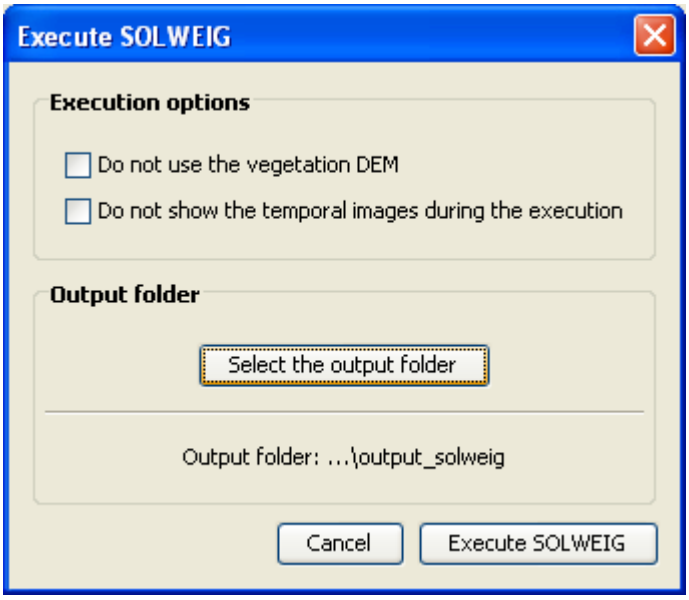

Figure 17: Execute SOLWEIG before executing the model

By clicking on the “Execute SOLWEIG” button (located at the right-bottom in Figure 17), the user will be able to simulate the first flow of the model.

See section 7.6 for a detailed description of this step from the system point of view.

4.8.8. Calculate Daily Shading

Figure 18 shows the execution dialog that is popped up when the button “Calculate Daily Shading” in the second flow is clicked (see Figure 8):

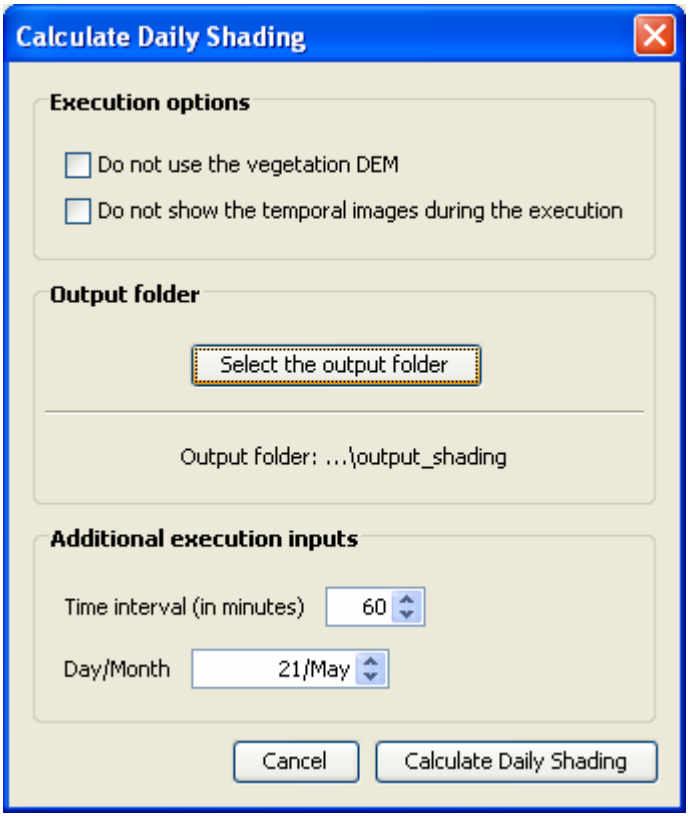

Figure 18: Calculate Daily Shading before executing the model

By clicking on the “Calculate Daily Shading” button (located at the right-bottom in Figure 18), the user will be able to simulate the second flow of the model.

See section 7.7 for a detailed description of this step from the system point of view.

5. Feedback of the testers

This section covers the validation of the system. It summarizes the different opinions and feedback that have been obtained by a group of seventeen testers.

The feedback is split in two groups:

- **Tasks feedback:** concerns questions related to a set of tasks that have been given to and done by the testers.
- **General feedback:** concerns questions related to the graphical interface.

The next sections will cover this feedback by listing the different questions answered by the testers and giving the results and comments obtained from each one.

5.1. Tasks feedback

This group of questions are related to a set of tasks that have been done by the testers. There are two types of tasks:

- **Interface-oriented tasks:** tasks one and two (see below). They try to measure whether the tasks can be done by just following some specified steps. Interpreting the results is not required; the aim is to check if the testers are able to manage all the steps by just dealing with the graphical interface. Anyone can do them.
- **Model-oriented tasks:** tasks three and four (see below). They try to measure whether the testers can answer the question that is asked. Interpreting the results is required; the aim is to demonstrate if the software is actually useful for those who might make use of it in the future. They are targeted to testers who know about the subject.

These tasks are defined below:

1) Execute the SOLWEIG model by:
    - Loading the building DEM.
    - Loading and editing the vegetation DEM. Add one tree of each type and save it in a new file.
    - Choosing Göteborg as the location.
    - Setting a point of interest.
    - Loading the building SVF.
    - Creating the vegetation SVF.
    - Not using the vegetation DEM in the final execution.

2) Execute the Calculate Daily Shading by:
    - Loading the building DEM.
    - Loading the vegetation DEM.
    - Choosing Göteborg as the location.
    - Setting a point of interest.
    - Not using the vegetation DEM in the final execution.

3) Model sensitivity: How does the mean radiant temperature (T_{mrt}) vary with altered “albedo” values in a courtyard?
4) Calculate the solar access at the first, second and third floor at the north wall in an east-west oriented street canyon 6 May.

Once the testers dealt with the different tasks, two questions were answered by them:

1) Could you manage to end your task? If not, why?
2) Did you find errors during the execution of your task? Where and when?

Despite of the fact of having two types of tasks, the conclusions and results will be shown by not taking into account the nature of the task. The reason of grouping them is the lack of a proper number of testers who have tested the model-oriented tasks and the fact that some testers did answer the questions from a general opinion after having tested all the tasks. Besides, the initial aim is to have a general idea of how the testers react when they deal with the interface. Future studies may pay special attention on the details that might cause the possible dissatisfaction of a user.

The results obtained from the answer of the two questions are shown in Figure 19 and Figure 20:

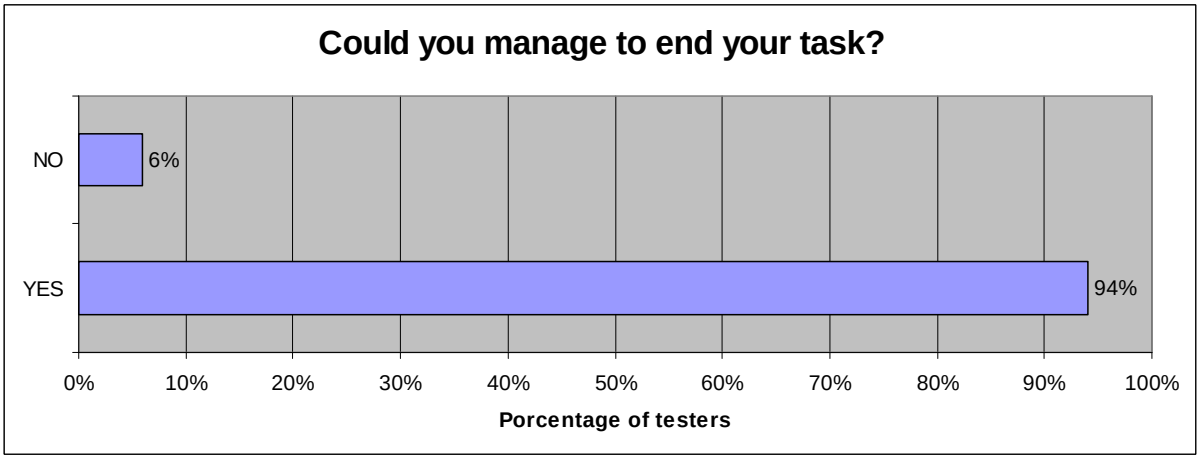

Figure 19: Tasks feedback – Could you manage to end your task?

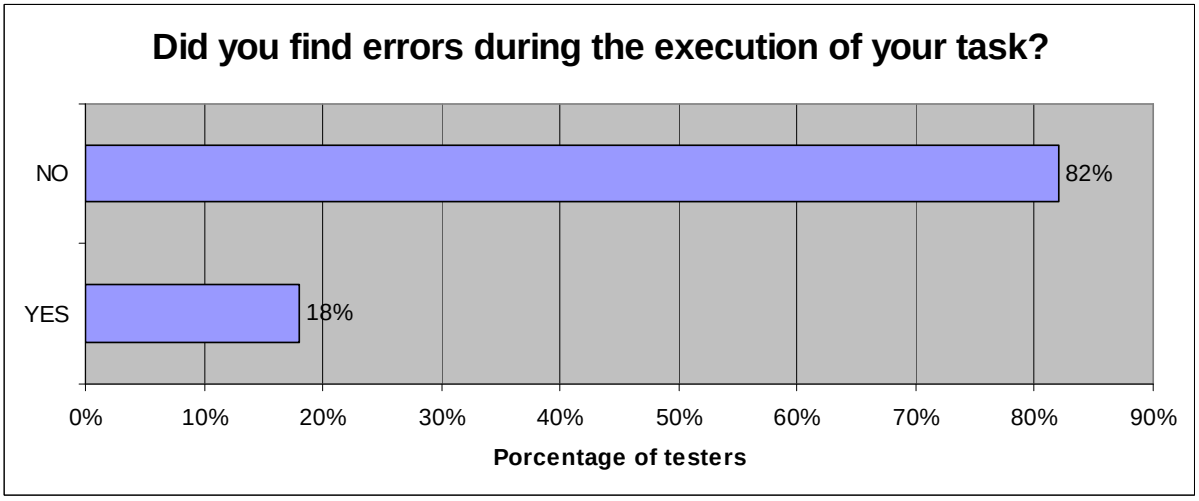

Figure 20: Tasks feedback – Did you find errors during the execution of your task?

As shown in Figure 19, the 94% of the testers could solve the tasks without having any problems. However, there were some cases where some errors where found (like Figure 20 indicates, the 18% of the testers found errors). Concretely in this case, these bugs were caused when the tester tried to load a file or type a specific input data. They are actually located and going to be solved.

5.2. General feedback

This feedback is defined by a set of questions that were asked to the testers once they tried the interface through the assigned tasks or by their own. They try to measure if the interface is user-friendly and if a future end-user would find it usable and would be satisfied with the behaviour of the software. The questions are listed below:

1) Was your assigned flow (Execute SOLWEIG or Calculate Daily Shading) easy to follow? Please evaluate on a scale from 1 (easy) to 7 (difficult).
2) Was the interface easy to use? Please evaluate on a scale from 1 (easy) to 7 (difficult).
3) Would you use the software more than once or is it too complicate or poor in terms of usability?
4) Please evaluate the interface by giving a mark between 1 (lowest mark) and 7 (highest mark).
5) Finally, write down your additional comments (optional).

The results for each question are shown below with the corresponding graphics:

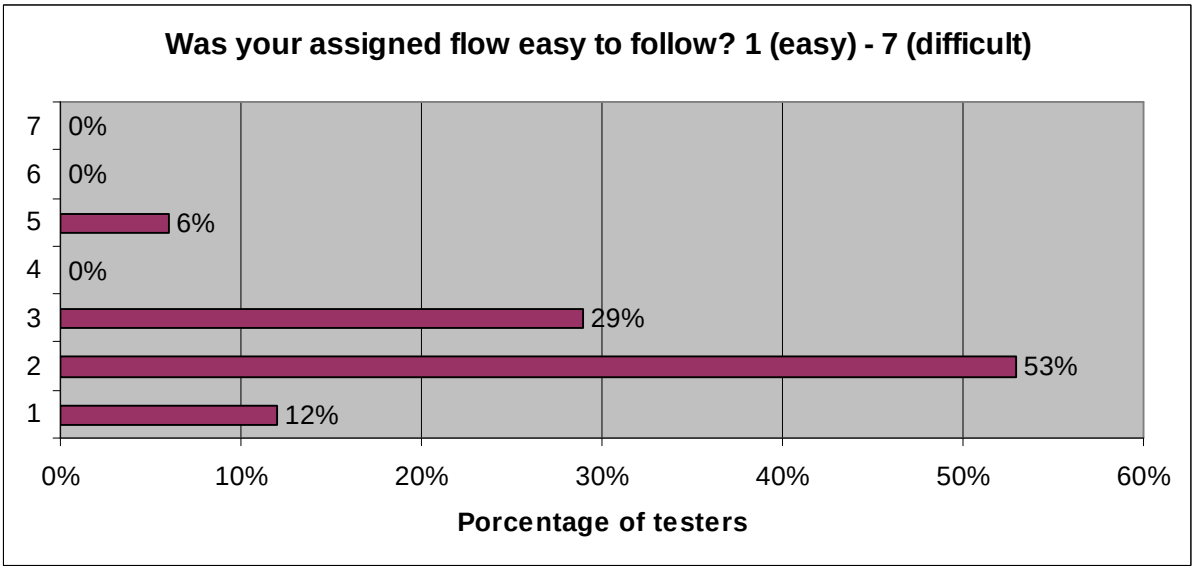

Figure 21: General feedback – Was your assigned flow easy to follow?

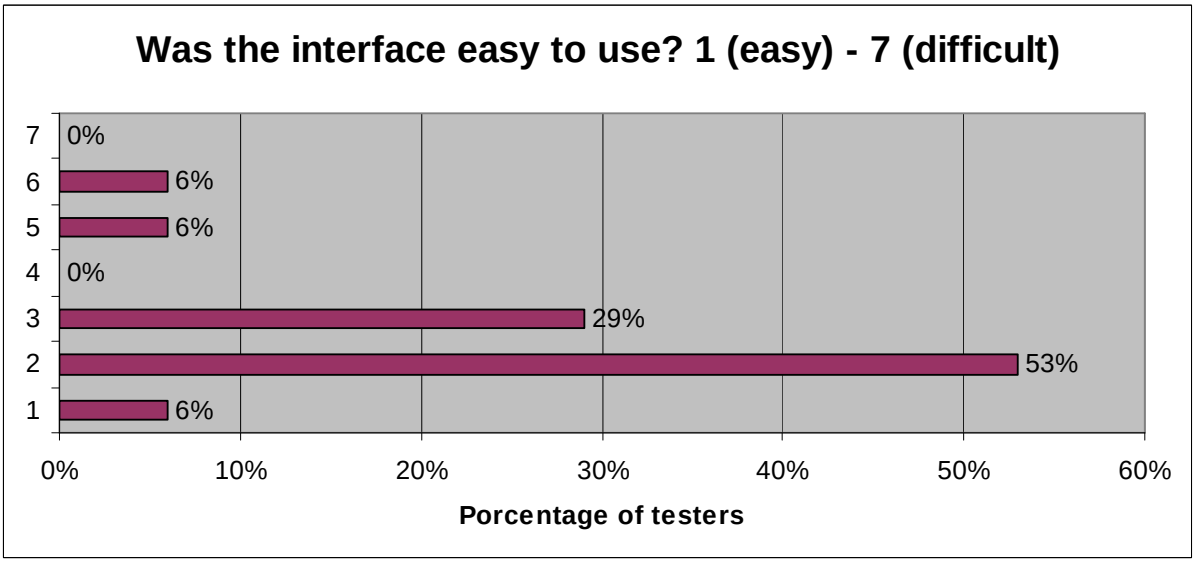

Figure 22: General feedback – Was the interface easy to use?

Figure 21 and Figure 22 presents the results for two questions that are quite related to the user-friendly requirement. By having a look at both figures, it can be concluded that on average the testers found the interface easy to follow and use (in both cases, the 53% of the testers gave a 2 as an answer, which means that they found it quite easy). However, there were some others that had some problems when trying to deal with it (for instance, the 6% of the testers gave a 6 in the second question, see Figure 22). Concretely, one of the testers had to give a try first in order to understand how to do the assigned task, while another one considers that there are many previous steps before getting any results. These two cases were studied and it was found out that both testers did not know about the subject and did not have a previous introduction to the model and interface, so they got lost especially when they tried to solve the model-oriented tasks.

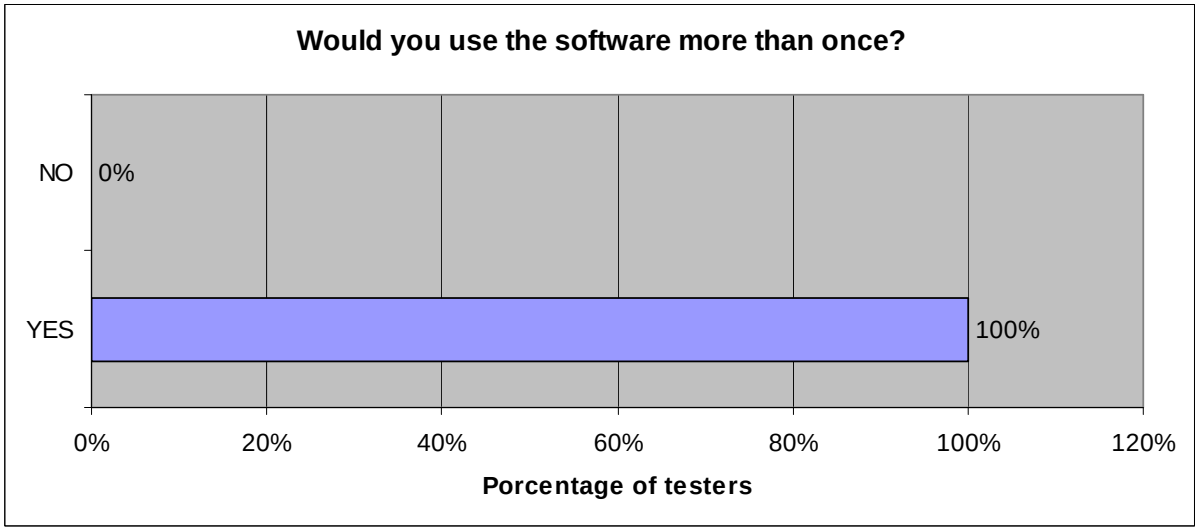

Figure 23: General feedback – Would you use the software more than once?

Figure 23 presents the results for a question that is related to the usability requirement. By having a look at the figure, it can be concluded as a success the fact that a future end-user would use the interface many times to get the desired results, as the 100% of the testers claimed that they would use the software again. In this way one of the most important targets of the project is more than fulfilled: usability.

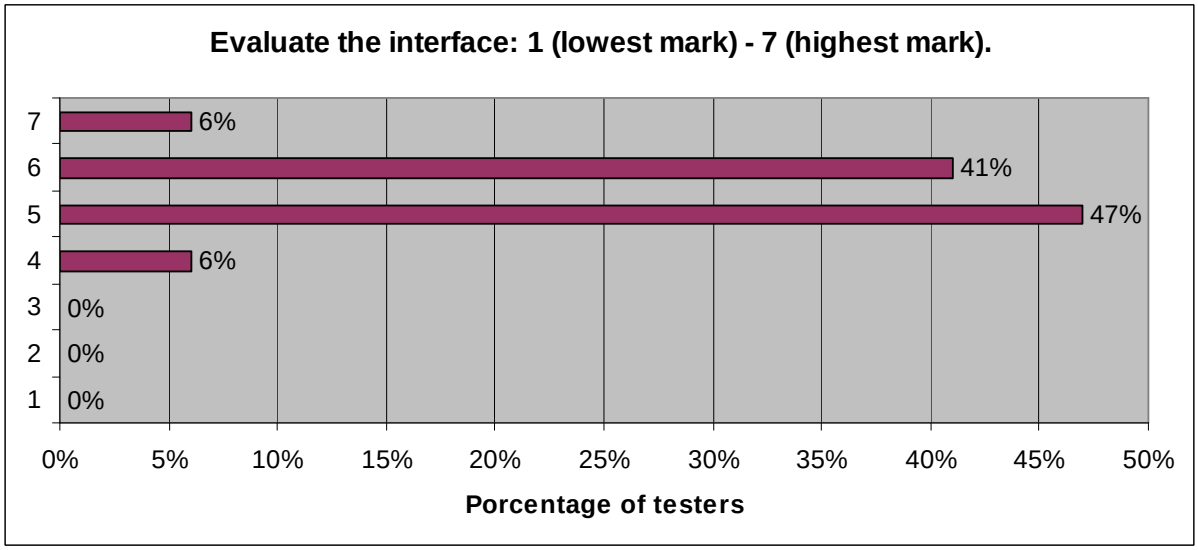

Figure 24: General feedback – Evaluate the interface

Figure 24 asks the testers to evaluate the interface. By having a look at the results provided in the figure, it can be concluded as very positive the final mark that the interface has received as a whole by the testers (the 88% of them marked it with a 5 or 6, which is a quite high mark). No negative marks have been given and that is a sign of satisfaction by the future end-user. In fact, and related to the last question, where the testers had the chance to express an opinion about the tool, good feedback was received as a summary by the testers. Concretely, some of them think that the interface is easy to use; another one likes the way it gets the results (nice-looking), which are quite easy to interpret; and another is satisfied with the tree that is provided to follow the actions and completions.

As a summary and by taking into account the previous results and opinions, the results obtained from the feedback can be concluded as satisfactory.

6. Conclusions

In this document, a graphical and user-friendly software interface for the SOLWEIG model has been presented. It solves the problem of sharing the model with other users by avoiding the restrictions that this software has. The tool is written in Java and uses the Swing library to show the graphical components.

Regarding the connection with the SOLWEIG model (which is written in MATLAB), the tools MATLAB Compiler and MATLAB Builder JA have been used to obtain a Java version of the model, so that the graphical interface can execute the MATLAB functions by making use of a MATLAB runtime engine called MATLAB Compiler Runtime. This application can be deployed royalty-free along with the graphical interface. Therefore the graphical interface can take advantage of MATLAB matrices processing and plots of DEMs in order to get the fastest and efficient simulation of the model; while, regarding the distribution of the application, it can be shared with researchers, architects and urban planners without the need of buying a license.

In order to develop the tool, a methodology has been followed. The main applied method has been the one concerning the meetings, with the researchers of the Urban Climate Group, from which the requirements of the graphical interface were obtained. These requirements have been: user-friendly interface, extensibility of the code and usability of the whole tool. They have been solved by developing and analysing technical documents that have defined the structure (both the external and the internal one) of the interface. It is worth to mention that a Model-View-Controller architecture has been applied, since it covers the extensibility requirement and facilitates the achievement of the other two ones.

Finally, the tool has been tested by a sample of seventeen testers who have given a first idea of how the tool impacts both a normal user and a researcher who is familiar to the SOLWEIG model. The results can be considered as satisfactory, as most of the questions answered by the testers have had a positive feedback. Besides, some comments and opinions have been taken into account for future versions of the software.

7. Appendixes – Scenarios

This section develops each use case that has been defined in section 4.4. In order to do this, each of them is going to be exploited in scenarios that will explain the interaction between the user and the graphical interface and, therefore, will represent the system requirements.

7.1. Load DEMs

Scenario 1: Basic scenario

1. The user selects the “Load DEMs” option.
2. The system shows the “Load DEMs” window, which initially has the following enabled options:
    - “Load building DEM” button: used to load the building DEM.
    - “Close” button: used to close the “Load DEMs” window.

Scenario 2: Button “Load building DEM” is pressed

1. The user presses the “Load building DEM” button.
2. The system shows a file chooser (any type of file extensions are allowed).
3. The user selects a building DEM file and presses the “Load” button from the file chooser.
4. The system closes the file chooser and checks whether the selected file corresponds with a valid building DEM (see section 8.1).

[If the building DEM is valid]

5. The system loads the building DEM and notifies this fact to the user. Besides, it hides the “Load building DEM” button and shows in the “Load DEMs” window new options:
    - “Load vegetation DEM” button: used to load the vegetation DEM.
    - “Create/Edit vegetation DEM” button: used to create/edit the vegetation DEM.
    - “Location” list: used to select the location the DEM belongs to. It is a list with the locations of the main cities in the world which is stored in two files within the system (see section 8.5).
    - “Add location” button: used to add new location within the system.
    - “Edit location” button: used to edit the parameters of the location selected in the “Choose location” list.
    - “Remove location” button: used to delete the selected location from the system.
    - “Load new DEMs” button: used to unload the current DEM/DEMs and start again in Scenario 1.

Scenario 3: Invalid building DEM format (from step 4 of Scenario 2)

[If the building DEM has not a valid format]

5. The system shows an “Invalid building DEM format” error message dialog. Types of errors:
    - There are more or less headers than expected.
    - The headers are not sorted in the corresponding order.
    - There is at least one incorrect value in the header.
    - The matrix has not the size specified in the headers.
6. The user presses the “OK” button from the current dialog.
7. The system closes the dialog.

Scenario 4: Button "Load vegetation DEM" is pressed

1. The user presses the "Load vegetation DEM" button.
2. The system shows a file chooser (any type of file extensions are allowed).
3. The user selects a vegetation DEM file and presses the "Load" button from the file chooser.
4. The system closes the file chooser and checks whether the selected file corresponds with a valid vegetation DEM (see section 8.6).

[If the vegetation DEM is valid]

5. The system loads the vegetation DEM and notifies this fact to the user.

Scenario 5: Invalid vegetation DEM (from step 4 of Scenario 4)

[If the building DEM is not valid]

5. The system shows an "Invalid vegetation DEM" error message dialog.
6. The user presses the "OK" button from the current dialog.
7. The system closes the dialog.

Scenario 6: Button "Create/Edit vegetation DEM" is pressed

1. The user presses the "Create/Edit vegetation DEM" button.

[If there is not a vegetation DEM file loaded]

2. The system shows the "Mark the buildings" window, which has the following options:
    - "Mark buildings" button: used to mark the buildings that appear within the building DEM. In order to do this, a MATLAB window shows the buildings to the user, who will have the chance to mark them as many times as needed.
    - "Undo last marking" button: used to undo the last marking made by the user (it is initially disabled and becomes enabled after having marked the first buildings). It is possible to undo one time, after this the undone action can be redone, so that the button will be renamed as "Redo last marking". Every time the user marks new buildings the button will be called and work as "Undo last marking".
    - "Continue" button: used to enter to the next step: "Edit vegetation DEM" window (the "Mark the buildings" window is automatically closed). This button is initially disabled and becomes enabled after having marked the first buildings.
    - "Cancel" button: used to cancel the whole action and to close the "Mark the buildings" window.

[In any case]

3. The system shows the "Edit vegetation DEM" window, which has the following options:
    - "Conifer" radio button: used to mark a conifer as the tree to be located.
    - "Deciduous" radio button: used to mark a deciduous as the tree to be located.
    - "Bush" radio button: used to mark a bush as the tree to be located.
    - "Diameter (m)" input: used to specify the diameter of the tree to be located (in meters). It is a decimal number from 0 to infinity.
    - "Tree height (m)" input: used to specify the height of the tree to be located (in meters). It is a decimal number from 0 to infinity.
    - "Trunk height (m)" input: used to specify the trunk size of the tree to be located (in meters). It is a decimal number from 0 to infinity. It cannot be equal or greater than the height of the tree. Besides, the bush tree will always have a value of 0.0 for this input.
    - "Locate" button: used to locate the selected tree within the building DEM. In order to do this, a MATLAB window with the building DEM been displayed on it is provided and activated to let the user click in the desired location over the DEM to locate the tree.
    - "Select the unit ID" combo box: used to choose the tree (through its identifier) that the user wants to remove.

- "Remove" button: used to remove the tree that has been selected within the "Select the unit ID" combo box.
- "Create a new vegetation DEM" button: used to restart the process and create a new vegetation DEM from the beginning, so that the "Mark the buildings" window is automatically shown and the system jumps to the step 2 of this scenario. Note: the system shows a warning dialog to the user before restarting the process, so that he/she has the chance to think before taking the decision.
- "Save" button: used to save the current situation of the vegetation. All the information regarding the trees is saved and will be part of a new vegetation DEM (see section 8.6). After this, the "Edit vegetation DEM" window is automatically closed.
- "Cancel" button: used to cancel the whole action and to close the "Edit vegetation DEM" window.

[If a vegetation DEM is loaded or created]

4. The system loads the vegetation DEM and notifies this fact to the user.

Scenario 7: Button "Add location" is pressed

1. The user presses the "Add location" button.
2. The system shows the "Add location" window, which has the following options:
   - "Country" input: used to type the name of the country.
   - "City" input: used to type the name of the city.
   - "Longitude" input: used to specify the longitude of the city. It is a positive or negative decimal number.
   - "Latitude" input: used to specify the latitude of the city. It is a positive or negative decimal number.
   - "Altitude" input: used to specify the altitude of the city. It is a positive or negative decimal number.
   - "UTC" combo box: used to specify the Coordinated Universal Time the city belongs to.
   - "Add" button: used to save the new location in the system and to close the "Add location" window. The "Choose location" list will be updated with the new location. See section 8.5.
   - "Cancel" button: used to cancel the whole action and to close the "Add location" window.

Scenario 8: Button "Edit location" is pressed

1. The user presses the "Edit location" button.
2. The system shows the "Edit location" window, which has the following options:
   - "Country" input: used to retype the name of the country. Initially it will have loaded the country of the selected location in the "Choose location" list.
   - "City" input: used to retype the name of the city. Initially it will have loaded the city of the selected location in the "Choose location" list.
   - "Longitude" input: used to specify the new longitude of the city. It is a positive or negative decimal number. Initially it will have loaded the longitude of the selected location in the "Choose location" list.
   - "Latitude" input: used to specify the latitude of the city. It is a positive or negative decimal number. Initially it will have loaded the latitude of the selected location in the "Choose location" list.
   - "Altitude" input: used to specify the altitude of the city. It is a positive or negative decimal number. Initially it will have loaded the altitude of the selected location in the "Choose location" list.

- “UTC” combo box: used to specify the Coordinated Universal Time the city belongs to. Initially it will have loaded the UTC of the selected location in the “Choose location” list.
- “Apply” button: used to save the new parameters of the location in the system and to close the “Edit location” window. The “Choose location” list will be updated with the modified location. See section 8.5.
- “Cancel” button: used to cancel the whole action and to close the “Edit location” window.

Scenario 9: Button “Remove location” is pressed

1. The user presses the “Remote location” button.
2. The system shows a “Remove location” question message dialog.

[If the user does not want to continue]

3. The system closes the dialog.

[If the user wants to continue]

3. The system closes the dialog, removes the location from the system and updates the “Choose location” list.

Scenario 10: Button “Load new DEMs” is pressed

1. The user presses the “Load new DEMs” button.
2. The system shows a “Load new DEMs” question message dialog.

[If the user does not want to continue]

3. The system closes the dialog.

[If the user wants to continue]

3. The system closes the dialog, unload the current DEM/DEMs and jumps to Scenario 1.

Scenario 11: Button “Close” is pressed

1. The user presses the “Close” button.
2. The system closes the “Load DEMs” window.

7.2. Set point of interest in the DEM

Scenario 1: Basic scenario

1. The user selects the “Set point of interest in the DEM” option.
2. The system shows the “Set point of interest” window, which initially has the following enabled options:
    - “x” input: where the system shows the value of the “x” coordinate where the point of interest is located within the DEM.
    - “y” input: where the system shows the value of the “y” coordinate where the point of interest is located within the DEM.
    - “Height” input: where the user has to specify a height from where the selected point is going to be studied (this height can be negative and decimal. The default value is 1.1).
    - “Set new point” button: used to set a new point of interest within the building DEM.
    - “Cancel” button: used to cancel the whole action and to close the “Set point of interest” window/s.

Scenario 2: Button “Set new point” is pressed

1. The user presses the “Set new point” button.
2. The system activates a MATLAB window where the user has the chance to choose a new point of interest within the current building DEM by clicking once with the mouse over the presented image. If the vegetation DEM is loaded, the system will show the vegetation units as well.
3. The user sets a point of interest by clicking on the DEM.
4. The system takes the coordinates and shows them to the user. Besides, it shows in the “Set point of interest” window a new option:
    - “Apply” button: used to set the point of interest and to close the “Set point of interest” window/s.

Scenario 3: Button “Cancel” is pressed

1. The user presses the “Cancel” button.
2. The system closes the “Set point of interest” window/s without setting any changes.

Scenario 4: Button “Apply” is pressed

1. The user presses the “Apply” button.
2. The system restores the height of the previous point of interest (if there was any), sets the new point of interest (and its height) and then closes the “Set point of interest” window/s. Besides, it shows in the “Set point of interest” window a new option:
    - “Clear current point” button: used to unset the current point of interest.

Scenario 5: Button “Clear current point” is pressed

1. The user presses the “Clear current point” button.
2. The system restores the height of the current point of interest, closes the “Set point of interest” window/s and jumps to Scenario 1.

7.3. Load-Create SVFs

Scenario 1: Basic scenario

1. The user selects the “Load-Create SVFs” option.
2. The system shows the “Load-Create SVFs” window, which has the following options:
    - “Load building SVF” button: used to load the building SVF.
    - “Create building SVF” button: used to create the five building SVFs for the loaded building DEM and save them in one compressed ZIP file.
    - “Load vegetation SVF” button: used to load the vegetation SVF. This button is disabled if the vegetation DEM is not loaded.
    - “Create vegetation SVF” button: used to create the five vegetation SVFs for the loaded vegetation DEM and save them in one compressed ZIP file. This button is disabled if the vegetation DEM is not loaded.
    - “Close” button: used to close the “Load-Create SVFs” window.

Scenario 2: Button “Load building SVF” or “Load vegetation SVF” is pressed

1. The user presses the “Load building SVF” or “Load vegetation SVF” button.
2. The system shows a file chooser that allows selecting ZIP files.
3. The user selects the ZIP file which contains the SVF (see section 8.3) and presses the “Load” button from the file chooser.
4. The system closes the file chooser, decompresses the ZIP file and checks whether the SVF in ZIP format is being followed (see section 8.3).

[If the SVF in ZIP format is valid]

5. The system checks whether each file corresponds with a valid SVF (see section 8.2).

[If the SVF is valid]

6. The system loads the SVF and notifies this fact to the user.

Scenario 3: Invalid SVF in ZIP format (from step 4 of Scenario 2)

[If the SVF in ZIP format is not valid]

5. The system shows an “Invalid SVF in ZIP format” error message dialog.
6. The user presses the “OK” button from the current dialog.
7. The system closes the dialog.

Scenario 4: Different SVF header (from step 5 of Scenario 2)

[If a SVF has not the same header than the building DEM]

6. The system shows a “Different SVF header” error message dialog.
7. The user presses the “OK” button from the current dialog.
8. The system closes the dialog.

Scenario 5: Button “Create building SVF” or “Create vegetation SVF” is pressed

1. The user presses the “Create building SVF” or “Create vegetation SVF” button.
2. The system shows a file chooser where the user has the chance to choose a folder and a name where to save the SVF in ZIP format (see sections 8.2 and 8.3).
3. The user chooses a destination path and presses the “Save” button from the file chooser.
4. The system closes the file chooser and creates the SVF, which is then saved in the corresponding folder. While its creation, a MATLAB progress bar is displayed to show the user the remaining time to finish the process.
5. The system loads the SVF and notifies this fact to the user.

Scenario 6: Button "Close" is pressed

1. The user presses the "Close" button.
2. The system closes the "Load-Create SVFs" window.

7.4. Set model parameters

Scenario 1: Basic scenario

1. The user selects the "Set model parameters" option.
2. The system shows the "Set model parameters" window, which has the following options:
   - The following list of parameters with their default values:

Urban parameters		
Parameter	**Description**	**Default value**
Albedo	Average albedo	0.15
Emissivity - wall	Emissivities of building walls	0.9
Emissivity - ground	Emissivities of the ground	0.95
Personal parameters		
Absorption of K	Absorption coefficient of short wave radiation of a person	0.7
Absorption of L	Absorption coefficient of long wave radiation of a person	0.97
Paste	The angular factors between a person and the surrounding surfaces	STAND
Fside	The angular factors between a person and the surrounding surfaces	0.22
Fup	The angular factors between a person and the surrounding surfaces	0.06

Table 1: Parameters description and default values

Note: only the coloured parameters are configurable. They are decimal numbers between 0 and 1 (including both).

- "Cancel" button: used to cancel the whole action and to close the "Set model parameters" window.
- "Apply" button: used to set the parameters and to close the "Set model parameters" window.

Scenario 2: Button "Cancel" is pressed

1. The user presses the "Cancel" button.
2. The system closes the "Set model parameters" window without setting any changes.

Scenario 3: Button "Apply" is pressed

1. The user presses the "Apply" button.
2. The system checks whether the values of the parameters are valid.

[If the values of the parameters are valid]

3. The system stores the new values. For the parameter "Paste" there are two possible options: "STAND" and "SIT"; depending on what the user has chosen, the "Fside" and "Fup" parameters would automatically get the following values:

PA value	Fside value	Fup value
STAND	0.22	0.06
SIT	0.166	0.166

Table 2: Value of the angular factors

4. The system closes the “Set model parameters” window.

Scenario 4: Invalid parameter values (from step 2 of Scenario 2)

[If the values of the parameters are not valid]
3. The system shows an “Invalid parameter values” error message dialog.
4. The user presses the “OK” button from the current dialog.
5. The system closes the dialog.

7.5. Add meteorological data

Scenario 1: Basic scenario

1. The user selects the “Add meteorological data” option.
2. The system shows the “Add meteorological data” window, which has the following options:
    - “Calculate direct and diffuse radiation from global radiation” check box: used to indicate whether the direct and diffuse radiation data should be obtained through the global radiation.
    - “Load meteorological data” button: used to load the meteorological data.
    - “Close” button: used to close the “Add meteorological data” window.

Scenario 2: Button “Load meteorological data” is pressed

1. The user presses the “Load meteorological data” button.
2. The system shows a file chooser (any type of file extensions are allowed).
3. The user selects a file and presses the “Load” button from the file chooser.
4. The system closes the file chooser and checks whether the selected file corresponds with a valid meteorological data file (see section 8.4).
[If the format is valid]
5. The system loads the meteorological data and notifies this fact to the user.

Scenario 3: Invalid meteorological data format (from step 4 of Scenario 2)

[If the file has not a valid format]
5. The system shows an “Invalid meteorological data format” error message dialog. Types of errors:
    - There are more or less columns than expected.
    - The columns are not sorted in the corresponding order.
    - There is at least one incorrect value in a certain row and column.
6. The user presses the “OK” button from the current dialog.
7. The system closes the dialog.

Scenario 4: Button “Close” is pressed

1. The user presses the “Close” button.
2. The system closes the “Add meteorological data” window.

7.6. Execute SOLWEIG

Scenario 1: Basic scenario

1. The user selects the “Execute SOLWEIG” option.
2. The system shows the “Execute SOLWEIG” window, which has the following options:
   - “Do not use the vegetation DEM” check box: used to indicate whether to use or not the vegetation DEM during the execution of the model. This option is enabled only if the vegetation DEM is loaded.
   - “Do not show the temporal images during the execution” check box: used to indicate whether to see or not the graphics shown during the execution.
   - “Select the output folder” button: used to select a folder where the output is stored.
   - “Cancel” button: used to cancel the whole action and to close the “Execute SOLWEIG” window.
   - “Execute SOLWEIG” button: used to execute the SOLWEIG model and to close the “Execute SOLWEIG” window.

Scenario 2: Button “Select the output folder” is pressed

1. The user presses the “Select the output folder” button.
2. The system shows a file chooser where the user has the chance to choose a folder where the output of the execution will take place.
3. The user chooses a destination path and presses the “Select” button from the file chooser.
4. The system closes the file chooser and notifies the output folder to the user.

Scenario 3: Button “Cancel” is pressed

1. The user presses the “Cancel” button.
2. The system closes the “Execute SOLWEIG” window.

Scenario 4: Button “Execute SOLWEIG” is pressed

1. The user presses the “Execute SOLWEIG” button.
2. The system checks if an output folder has been specified.

[If an output folder has been specified]

3. The system closes the “Execute SOLWEIG” window and executes the SOLWEIG model. A MATLAB window is shown at the end of the process with a summary of the data that has been stored in the output folder.

Scenario 5: Invalid output folder (from step 2 of Scenario 4)

[If an output folder has not been specified]

3. The system shows an “Invalid output folder” error message dialog.
4. The user presses the “OK” button from the current dialog.
5. The system closes the dialog.

7.7. Calculate Daily Shading

Scenario 1: Basic scenario

1. The user selects the “Calculate Daily Shading” option.
2. The system shows the “Calculate Daily Shading” window, which has the following options:
   - “Do not use the vegetation DEM” check box: used to indicate whether to use or not the vegetation DEM during the execution of the model. This option is enabled only if the vegetation DEM is loaded.
   - “Do not show the temporal images during the execution” check box: used to indicate whether to see or not the graphics shown during the execution.
   - “Select the output folder” button: used to select a folder where the output is stored.
   - “Execution interval” spinner: where the user has to specify the time interval the model has to base its execution (this interval is specified in minutes, where the minimum value is 1 and the default is 60 minutes).
   - “Day/Month” spinner: where the user has to specify the day and the month the model has to base its execution.
   - “Cancel” button: used to cancel the whole action and to close the “Calculate Daily Shading” window.
   - “Calculate Daily Shading” button: used to calculate the daily shading and to close the “Calculate Daily Shading” window.

   Note: the combination “Day = 29” and “Month = 2” is not allowed by the system.

Scenario 2: Button “Select the output folder” is pressed

1. The user presses the “Select the output folder” button.
2. The system shows a file chooser where the user has the chance to choose a folder where the output of the execution will take place.
3. The user chooses a destination path and presses the “Select” button from the file chooser.
4. The system closes the file chooser and notifies the output folder to the user.

Scenario 3: Button “Cancel” is pressed

1. The user presses the “Cancel” button.
2. The system closes the “Calculate Daily Shading” window.

Scenario 4: Button “Calculate Daily Shading” is pressed

1. The user presses the “Calculate Daily Shading” button.
2. The system checks if an output folder has been specified.

[If an output folder has been specified]

3. The system closes the “Calculate Daily Shading” window and calculates the daily shading. A MATLAB window is shown at the end of the process with a summary of the data that has been stored in the output folder.

Scenario 5: Invalid output folder (from step 2 of Scenario 4)

[If an output folder has not been specified]

3. The system shows an “Invalid output folder” error message dialog.
4. The user presses the “OK” button from the current dialog.
5. The system closes the dialog.

8. Appendixes – Other information

This section describes the different file formats that the graphical interface follows in order to load the input data that must be loaded from a file.

8.1. Building DEM format

This section explains the format that a building DEM has to follow in order to be loaded in the system.

By default, the system will allow all types of file extensions in where a building DEM can be stored. In order to be successfully loaded, it has to follow the following format (including the order of the headers):

ncols?#	*(# = a round number greater than zero = number of columns of the matrix)*
nrows?#	*(# = a round number greater than zero = number of rows of the matrix)*
xllcorner?#	*(# = a positive or negative decimal number = geographic "x" coordinate of the lower corner of the matrix)*
yllcorner?#	*(# = a positive or negative decimal number = geographic "y" coordinate of the left side of the matrix)*
cellsize?#	*(# = a positive decimal number, from 0 = size of 1 pixel)*
NODATA_value?#	*(# = a positive or negative decimal number = the value of no data)*

The matrix of positive and/or negative decimal numbers representing the DEM.
Each row is separated by a new line and each column by a blank character.
The size is the one specified in the "ncols" and "nrows" headers.

Note: *(? = 1 or more blank characters, including tabs).*

An example of the above building DEM format is shown below:

ncols 350
nrows 350
xllcorner 39250
yllcorner 27993
cellsize 1
NODATA_value -9999
0.723 0.207 0.341 0.408 0.439 0.455 0.463 0.461 0.445 0.409 0.371 0.36 0.347
0.337 0.319 0.312 0.312 0.301 0.297 0.294 0.289 0.285 0.276 0.275 0.268 0.257
0.244 0.199 0.924 0.924 0.923 0.928 0.924 0.931 0.931 0.934 0.935 0.937 0.939
0.942 0.941 0.944 0.949 0.953 0.954 0.969 0.976 0.977 0.301 0.461 0.341 0.337

8.2. SVF format

This section explains the format that the SVF (represented in five files) have to follow in order to be loaded in the system. Each of the files must have the same format.

By default, the system will allow all types of file extensions in where a SVF can be stored (the system will use the ASC extension when it has to create a SVF by its own, as it is an ASCII file). In order to be successfully loaded, it has to follow the following format (including the order of the headers):

ncols?#	*(# = a round number greater than zero = number of columns of the matrix)*
nrows?#	*(# = a round number greater than zero = number of rows of the matrix)*
xllcorner?#	*(# = a positive or negative decimal number = geographic "x" coordinate of the lower corner of the matrix)*
yllcorner?#	*(# = a positive or negative decimal number = geographic "y" coordinate of the left side of the matrix)*
cellsize?#	*(# = a positive decimal number, from 0 = size of 1 pixel)*
NODATA_value?#	*(# = a positive or negative decimal number = the value of no data)*

The matrix of positive and decimal numbers between 0 and 1 (including both) representing the SVF. Each row is separated by a new line and each column by a blank character. The size is the one specified in the "ncols" and "nrows" headers.

Note: *(? = 1 or more blank characters, including tabs).*

An example of the above SVF format is shown below:

ncols 350
nrows 350
xllcorner 39250
yllcorner 27993
cellsize 1
NODATA_value -9999
0.723 0.207 0.341 0.408 0.439 0.455 0.463 0.461 0.445 0.409 0.371 0.36 0.347
0.337 0.319 0.312 0.312 0.301 0.297 0.294 0.289 0.285 0.276 0.275 0.268 0.257
0.244 0.199 0.924 0.924 0.923 0.928 0.924 0.931 0.931 0.934 0.935 0.937 0.939
0.942 0.941 0.944 0.949 0.953 0.954 0.969 0.976 0.977 0.301 0.461 0.341 0.337

Important: both the building and the vegetation SVFs must have the same header (that is, the same values for the first five lines) and size of the matrix than the building DEM.

8.3. SVF in ZIP

This section explains the format that the SVF have to follow in order to be loaded in the system from a ZIP file. In order to do this, an illustrative example is shown below:

If the ZIP file is called, for instance, test.zip, then the SVF must have the following names (the extension is irrelevant):

- test
- testN
- testS
- testE
- testW

Each file that represents the SVF should then follow the format explained in section 8.2.

The system will follow the same format when creating and saving the SVF by its own. This means that both the ZIP file and the five files that represent the SVF will make use of the name given by the user. Besides, the five files of the SVF will be also added in their names the corresponding directions that they represent (that is: N, S, E, W or nothing for all directions).

8.4. Meteorological data format

This section explains the format that the meteorological data file has to follow in order to be able to load the meteorological data in the system.

By default, the system will allow all types of file extensions in where the meteorological data can be stored. In order to be successfully loaded, it has to follow the following format (including the order of the columns):

year	*month*	*day*	*hour*	*Ta*	*RH*	*radG*	*radD*	*radI*
yyyy	*mm*	*dd*	*h*	*a*	*b*	*c*	*d*	*e*

Where all the columns are separated by a tab and:

- yyyy = a year with 4 digits.
- mm = a month (a round number between 1 and 12, including both).
- dd = a day (a round number between 1 and 31, including both. The values 29, 30 and 31 can appear depending on the chosen month and year, as it is specified in the Gregorian calendar).
- h = an hour (a round number between 0 and 23, including both).
- a = the air temperature (a positive or negative decimal number).
- b = the relative humidity (a decimal number between 0 and 100, including both).
- c = the global shortwave radiation (a decimal number from -50 to infinity).
- d = the diffuse shortwave radiation (a decimal number from -50 to infinity).
- e = the direct shortwave radiation (a decimal number from -50 to infinity).

An example of the above format is shown below:

year	*month*	*day*	*hour*	*Ta*	*RH*	*radG*	*radD*	*radI*
2005	*10*	*11*	*1*	*12.9*	*92*	*0*	*0*	*0*
2005	*10*	*11*	*2*	*12.6*	*92*	*0*	*0*	*0*
2005	*10*	*11*	*7*	*11.9*	*89*	*5.1*	*5.1*	*0*
2005	*10*	*11*	*8*	*12.5*	*86*	*63.2*	*39.3*	*222.1*
2005	*10*	*11*	*9*	*14.3*	*78*	*172*	*59.8*	*499.1*
2005	*10*	*11*	*11*	*17.5*	*64*	*347.2*	*75.5*	*695.7*

8.5. Location files

This section explains all the information relative to the location files. First, the idea behind the location files will be explained and then the way the system deals with them will be detailed.

As an effort of saving time and being able to achieving a user-friendly interface, the system has a small storage where the most important locations in the world are presented. The information regarded for a location is the one shown below:

- **Longitude**.
- **Latitude**.
- **Altitude**.
- **UTC**.

This information is always needed for the system to be executed, as every DEM belongs to a certain city and the weather conditions change depending on where the place is located around the world. Therefore it is necessary to situate the map in order to obtain reliable results.

The system will store a basic list of the main locations in one file, which is going to be considered as a global file. The format of this file is shown below:

name_country name_city longitude latitude altitude utc

Where all the columns are separated by a tab and:

- name_country = the name of the country (for instance: Sweden).
- name_city = the name of the city (for instance: Gothenburg).
- longitude = the longitude where the city is located.
- latitude = the latitude where the city is located.
- altitude = the altitude where the city is located.
- utc = the UTC of the city.

Besides there is another file, the so called local file, which is going to be empty at the beginning and is going to be the one where there will be stored the different locations the user might add.

Both the global and the local files have the same format and will be always sorted by the name of the country and city (for each country). The user will never realize the difference of these two files, as he/she will always see one sorted list (which will contain both lists together).

When a new version of the software comes, a new version of the global file might be published too, while the local file will be empty again, but in this case the user can replace the local one with the one he/she has from the previous version of the software, so that he/she can keep the locations he/she added and avoid re-adding them again.

If the user modifies a location that belongs to the global file, then the system will move this location to the local file and with the new values of the parameters. When a new global file is downloaded, then the user will see two entries with the same name of the city, so that he/she will have to remove the one that is not required (probably the one from the global file).

8.6. Vegetation DEM format

This section explains the format that a vegetation DEM has to follow in order to be loaded in the system.

By default, the system will allow all types of file extensions in where a vegetation DEM can be stored. In order to be successfully loaded, it has to follow the following format (including the order of the headers):

ID	*ttype*	*dia*	*height*	*trunk*	*x*	*y*	*build*
i	*t*	*d*	*h*	*tr*	*x*	*y*	*b*

Where all the columns are separated by a tab and:

- *i = tree identifier (a round number from 1 to infinity).*
- *t = tree type (a round number that can only have the three following values: 1 = Conifer; 2 = Deciduous; 3 = Bush).*
- *d = tree diameter in meters (a decimal number from 0 to infinity).*
- *h = tree height in meters (a decimal number from 0 to infinity).*
- *tr = tree trunk size in meters (a decimal number from 0 to infinity). This value cannot be equal or greater than the tree height. Besides, the bush tree will always have a value of 0.0 for this column.*
- *x = 'x' coordinate from the building DEM where the tree is located (a round number from 1 to the maximum 'x' value of the building DEM).*
- *y = 'y' coordinate from the building DEM where the tree is located (a round number from 1 to the maximum 'y' value of the building DEM).*
- *b = an area that corresponds with a marked building from the building DEM. This value is automatically assigned by the application the first time the user marks the buildings. Therefore if new trees are added manually, this value has to be 0.0 (decimal format). On the contrary, if there are marked buildings but not trees, there will be entries with values 0.0 in all the columns excepting in the "build" one.*

An example of the above vegetation DEM format is shown below:

ID	*ttype*	*dia*	*height*	*trunk*	*x*	*y*	*build*
0.0	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*16873.0*
2.0	*1.0*	*10.0*	*30.0*	*5.0*	*128.0*	*133.0*	*17307.0*
3.0	*3.0*	*5.0*	*5.0*	*0.0*	*182.0*	*58.0*	*10155.0*
4.0	*2.0*	*15.0*	*20.0*	*5.0*	*133.0*	*40.0*	*23081.0*
5.0	*1.0*	*5.0*	*6.0*	*5.0*	*144.0*	*234.0*	*19425.0*

Important: every time a new vegetation file is saved (or loaded) within the interface, a new vegetation SVF must be created (or loaded) as well.

9. Definitions, acronyms and abbreviations

This section provides useful information for the reader while the document is being read, such as a list with the definition of uncommon or double meaning words and the acronyms and abbreviations used along the document.

9.1. Definitions

.ASC file extension: It is an ASCII text file.

Button: It is a part of a window that triggers a sequence of actions, when it is pressed, like opening a new window or storing some information.

Input data: It refers to all the main files and information that must be indicated by the user in order to execute the SOLWEIG model.

Software: Is a general term used to describe a collection of computer programs, procedures and documentation that perform some task on a computer system [6].

System: It is the final application itself, which interacts with the user and allows him/her to execute the offered services.

ZIP file format: It is a data compression and archive format. A ZIP file contains one or more files that have been compressed to reduce file size, or stored as-is.

9.2. Acronyms and abbreviations

ASCII: American Standard Code for Information Interchange.

DEM: Digital Elevation Model [4].

MCR: MATLAB Compiler Runtime.

SOLWEIG: SOlar and LongWave Environmental Irradiance Geometry.

SRS: Software Requirements Specification [8].

SVF: Sky View Factor.

UTC: Coordinated Universal Time.

URS: User Requirements Specification [7].

10. References

[1] Dia: http://live.gnome.org/Dia
Last verification: January, 29th of 2009.

[2] ZIP file format:
http://en.wikipedia.org/wiki/ZIP_(file_format)#Implementing_a_ZIP_application
Last verification: February, 11th of 2009.

[3] Original paper: SOLWEIG 1.0 – Modelling spatial variations of 3D radiant fluxes and mean radiant temperature in complex urban settings.
Fredrik Lindberg, Björn Holmer and Sofia Thorsson.

[4] DEM (Digital Elevation Model): http://en.wikipedia.org/wiki/Digital_elevation_model
Last verification: May, 18th of 2009.

[5] MATLAB programming language: http://en.wikipedia.org/wiki/Matlab
Last verification: May, 18th of 2009.

[6] Software: http://en.wikipedia.org/wiki/Computer_software
Last verification: May, 18th of 2009.

[7] URD: http://en.wikipedia.org/wiki/User_requirements_document
Last verification: May, 19th of 2009.

[8] SRS: http://en.wikipedia.org/wiki/Software_Requirements_Specification
Last verification: May, 19th of 2009.

[9] MathWorks: http://www.mathworks.com/
Last verification: May, 19th of 2009.

[10] Freeware: http://en.wikipedia.org/wiki/Freeware
Last verification: May, 19th of 2009.

[11] Java programming language:
http://en.wikipedia.org/wiki/Java_(programming_language)
Last verification: May, 19th of 2009.

[12] Java Swing library: http://en.wikipedia.org/wiki/Swing_(Java)
Last verification: May, 19th of 2009.

[13] Visual Basic .NET programming language:
http://en.wikipedia.org/wiki/Visual_Basic_.NET
Last verification: May, 19th of 2009.

[14] C++ programming language: http://en.wikipedia.org/wiki/C%2B%2B
Last verification: May, 19th of 2009.

[15] Object oriented programming: http://en.wikipedia.org/wiki/Object_oriented
Last verification: May, 19th of 2009.

[16] MATLAB Builder JA: http://www.mathworks.com/products/javabuilder/
Last verification: May, 20th of 2009.

[17] NetBeans IDE 6.5: http://www.netbeans.org/
Last verification: May, 25th of 2009.

[18] Comparison of Java and C++:
http://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B
Last verification: June, 18th of 2009.